Burroughs

Programmer's Guide

A Series DMSII Transaction Processing System (TPS)

(Relative to the Mark 3.5 System Software Release)

Distribution Code SE

Priced Item
Printed in U.S.A.
December 1984

1164043

Burroughs

Programmer's Guide

A Series DMSII Transaction Processing System (TPS)

(Relative to the Mark 3.5 System Software Release)

Copyright © 1984 Burroughs Corporation, Detroit, Michigan 48232

Distribution Code SE

Priced Item
Printed in U.S.A.
December 1984

1164043

SEP 25 1986

Burroughs

PUBLICATION CHANGE NOTICE

PCN No.: 1164043-001 Date: 10-24-85

Publication Title: A Series DMSII Transaction Processing System (TPS) Programmer's Guide PCN-001 (Relative to the Mark 3.6 System Software Release)

Other Affected Publications: None

Supersedes: N/A

Description:

This PCN presents revisions and additions to the DMSII Transaction Processing System (TPS) Programmer's Guide (Relative to the Mark 3.6 System Software Release). Revisions to the text are indicated by vertical black lines on the affected pages.

Replace These Pages	Add These Pages
~~Contents~~	6A - 6B
~~5~~	16A - 16B
~~15~~	80A - 80B
~~29~~	166A - 166B
~~79~~	194A - 194B
~~113~~	204A - 204B
~~115~~	234A - 234B
~~165~~	254A - 254B
~~175~~	258A - 258B
~~177~~	314A - 314B
~~193~~	315 - 324
~~203~~	
~~207~~	
~~219~~	
~~221~~	
~~233~~	
~~241~~	
~~253~~	
~~257~~	
~~313~~	

Retain this PCN cover sheet as a record of changes made to the basic publication.

COPYRIGHT c 1984, 1985
BURROUGHS CORPORATION
Detroit, Michigan 48232

Burroughs cannot accept any financial or other responsibilities that may be the result of your use of this information or software material, including direct, indirect, special or consequential damages. There are no warranties extended or granted by this document or software material.

You should be very careful to ensure that the use of this software material and/or information complies with the laws, rules, and regulations of the jurisdictions with respect to which it is used.

The information contained herein is subject to change without notice. Revisions may be issued to advise of such changes and/or additions.

Comments or suggestions regarding this document should be submitted on a Field Communication Form (FCF) with the Class specified as "2" (System Software), the Type specified as "1" (F.T.R.), and the Product specified as the seven-digit form number of the manual. The FCF should be sent to the following address:

Burroughs Corporation
PA&S/Orange County
19 Morgan
Irvine, CA 92718

Burroughs cannot accept any financial or other responsibilities that may be the result of your use of this information or software material, including direct, indirect, special or consequential damages. There are no warranties extended or granted by this document or software material.

You should be very careful to ensure that the use of this software material and/or information complies with the laws, rules, and regulations of the jurisdictions with respect to which it is used.

The information contained herein is subject to change without notice. Revisions may be issued to advise of such changes and/or additions.

Comments or suggestions regarding this document should be submitted on a Field Communication Form (FCF) with the Class specified as "2" (System Software), the Type specified as "1" (F.T.R.), and the Product specified as the seven-digit form number of the manual (for example, "1164043"). The FCF should be sent to the following address:

Burroughs Corporation
Product Assurance and Support
19 Morgan
Lake Forest, CA. 92718

CONTENTS

1 INTRODUCTION 1
SCOPE OF MANUAL. 2
ORGANIZATION OF MANUAL 2
RELATED DOCUMENTS. 5

2 OVERVIEW 6A
TRANSACTIONS AND TRANSACTION RECORDS 7
UPDATE LIBRARY 9
TRANSACTION LIBRARY. 9
TRANSACTION JOURNALS 10
REMOTE LIBRARY 12
HOSTINTERFACE PROGRAM. 12
TPS SOFTWARE COMPONENTS. 14

3 TRANSACTION FORMATTING LANGUAGE (TFL). 16A
TFL BASIC CONSTRUCTS 17
Character Set. 17
Digit. 18
Letter 18
Comment. 18
Family Name. 19
Simple Identifier. 19
Integer 20
Number. 21
Remark. 21
String. 22
Expression. 22
TFL DESCRIPTION OF TRANSACTION BASE. 28
Parameters 30
Defaults 33
Transaction Record Format. 37
Transaction Subbase. 41
Transaction Journal. 44
Items. 49
APPEARANCE OF TFL ITEMS TO APPLICATION LANGUAGES 64
UPDATING TRANSACTION BASES 66
SYSTEM/TFL 68
SYSTEM/TFL Input Files 68
SYSTEM/TFL Dollar-Card Options 69

4 USER LANGUAGE INTERFACE TO TPS 80A
COBOL/COBOL74 USER LANGUAGE INTERFACE TO TPS 80A
Invocation of Transaction Base 80B
Declaration of Transaction Record Variables. 84
Creation of Transaction Records. 86
Accessing Transaction Record Items 88
Transaction Record Control Items 93
Transaction Compile-Time Functions 100
Transaction Record Variables as Parameters 104
COBOL74 Specific Constructs for Parameter Passing. . . . 105
Transaction Record Variable Assignment 107
Transaction Use Procedures 108
Transaction Record Use Restrictions. 109

Compiler Checks. 110
Sample User-Written Program. 110
ALGOL INTERFACE TO TPS 113
Invocation of Transaction Base 113
Declaration of Transaction Record Variables. 117
Creation of Transaction Records. 119
Accessing Transaction Record Items 121
Transaction Record Control Items 125
Transaction Compile-Time Functions 131
Transaction Record Variables as Parameters 134
Transaction Record Variable Assignment 134
Transaction Record Use Restrictions. 136
Compiler Checks. 136
Sample User-Written Program. 136

5 TRANSACTION LIBRARY. 139
INTRODUCTION 139
COMPILING THE TRANSACTION LIBRARY. 139
LIBRARY INITIATION 142
DEFINITION OF TRANSACTION USER 148
ENTRY POINTS 150
TRANSACTION LIBRARY DIAGNOSTIC MESSAGES. 160
TPS STATISTICS 160

6 UPDATE LIBRARY 166A
PROGRAMMING THE UPDATE LIBRARY 166A
Parameter Passing to a COBOL74 Update Library. 168
Methods of Structuring the Update Library. 169
Example of Update Library. 177
DMSII USER LANGUAGE SYNTAX FOR TRANSACTION PROCESSING. . . 180
BEGINTRANSACTION, MIDTRANSACTION, and ENDTRANSACTION . . 180
OPEN Statement TRUPDATE Option 183
Data Base Consistency And Update Library Programming . . 184
REPRODUCIBILITY. 191

7 REMOTE LIBRARY AND HOSTINTERFACE 194A
COMPILING THE REMOTE LIBRARY 194A
HOW THE REMOTE LIBRARY WORKS 197
ERROR CONDITIONS 199
SAMPLE REMOTE LIBRARY STACK CONFIGURATION. 201

8 EXCEPTION HANDLING 203
EXCEPTION REPORTING. 203
Fatal Errors 203
Exceptions 208
CATEGORY 1 : 001 - 099 (ATTENTION) 208
CATEGORY 2 : 100 - 199 (REMOTE-HOST INTERFACE) 209
CATEGORY 3 : 200 - 209 (BAD PARAMETER VALUES). 209
CATEGORY 4 : 300 - 399 (FUNCTION NOT PERMITTED). 211
CATEGORY 5 : 400 - 499 (TRANSACTION REJECTED). 213
CATEGORY 6 : 500 - 599 (TRANSACTION FAILED). 215
CATEGORY 7 : 900 - 999 (SOFTWARE/HARDWARE/PROCEDURAL ERRORS 216
DMS EXCEPTION HANDLING 218
EXCEPTION HANDLING MECHANISM 222

9 SECURITY 223

10 TPS RECOVERY 225
TRANSACTION RECOVERY DESIGN FEATURES 227
TRANSACTION RECOVERY AFTER A HALT/LOAD 229
TRANSACTION RECOVERY AFTER AN ABORT. 230
JOURNAL ROLLBACK 231
REINITIALIZING AN EXISTING JOURNAL 232
Case 1. REBUILD 232
Case 2. ROLLBACK. 232
RESTARTING OF USER-WRITTEN PROGRAMS. 233

11 TRUTILITY. 234A
COMPILING TRUTILITY. 234A
RUNNING TRUTILITY. 237
TRUTILITY STATEMENTS 238
INITIALIZE Statement 239
SEARCH Statement 240
RECOVERCF Statement. 247
CONTROLINFO Statement. 250

12 CONVERTING TO TPS. 251
CONVERTING TO MIXED MODE 252
MIXED MODE RESTRICTIONS THAT ENSURE REPRODUCIBILITY. . . . 252

A HOW TO READ RAILROAD DIAGRAMS. 254A
Railroad Components. 255

B GLOSSARY 258A

C EXAMPLE OF A TRANSACTION BASE. 275
DASDL FOR BANKING EXAMPLE: 275
TFL DESCRIPTION OF TRANSACTION BASE. 276
EXAMPLES OF USER-WRITTEN PROGRAMS. 279
COBOL. 279
ALGOL. 290
EXAMPLES OF UPDATE LIBRARY 299
COBOL. 299
ALGOL. 303
EXAMPLE OF DETANKING PROCEDURE 308
ALGOL. 308

D SYSTEMTR TRANSACTION FORMAT. 314A

INDEX

1. INTRODUCTION

For users of Data Management System II (DMSII) software, the Transaction Processing System (TPS) provides improved methods for processing a high volume of transactions. TPS separates into modules the various functions needed to perform data base processing and provides the user with a library of transaction processing procedures, thereby minimizing the program coding and maintenance required of the user.

With TPS, coding programs that perform on-line collection of input and output data for specific transactions is less complicated than in a conventional DMSII environment. Programmers who write this kind of program need not understand the structure of the data base, nor how to program for interruptions in processing. Only one TPS module accesses the data base, so that changes to the structure of the data base usually do not affect other TPS modules.

TPS keeps track of all input transactions that access the data base. If an interruption in processing occurs, TPS determines which transactions have been processed against the data base and which have not, and automatically resubmits transactions that have not yet been processed.

In addition, TPS enables the user to batch data for later processing, and to perform processing on a data base that resides in a remote system. The TPS user can submit transactions to a centralized data base from any number of remote systems in a data communications network, without having to rewrite any TPS programs.

The main advantages of utilizing TPS can be summarized as follows:

1. TPS minimizes the amount of program coding and maintenance that the user must undertake.

2. TPS eliminates much of the complexity that normally afflicts programming for data base processing.

3. TPS provides a central definition of all transactions to be performed against a data base.

4. TPS provides comprehensive recovery capabilities.

5. TPS helps reduce the user's expenses in a number of other ways that are explained in later sections of this manual.

SCOPE OF MANUAL

This manual is a programming guide intended primarily for experienced large systems programmers who are utilizing TPS with their DMSII software. However, parts of the manual are useful to programmers who write programs that collect input and output data for specific transactions. Though not a tutorial text, this manual gives details and examples for all essential areas of TPS use.

ORGANIZATION OF MANUAL

This manual discusses each module of TPS and related issues. The most basic aspects of TPS are introduced first so that the user is familiar with TPS terminology and components by the time the more complex aspects are discussed.

Section 1 Introduction

This section discusses the overall uses and features of TPS and the scope and organization of the manual. Instructions on how to read syntax diagrams and a description of the heading-level format used in the manual are included.

Section 2 Overview

This section introduces the most basic components of TPS and explains how they interact. Components discussed include transactions and transaction records, the Update Library, the Transaction Library, transaction journals, the Remote Library, and the HOSTINTERFACE program.

Section 3 Transaction Formatting Language (TFL)

This section provides syntax and semantics of the Transaction Formatting Language used to define transaction record formats. This section also discusses the TFL processor, which creates a description file of all transaction record formats, and describes the processor files and processor options.

Section 4 User Language Interface To TPS

This section gives syntax and semantics for TPS extensions to the interface languages (ALGOL and COBOL/COBOL74), as well as skeletal example programs in the interface languages.

Section 5 Transaction Library

This section explains the functions of the Transaction Library, also known as the Host Library, that is tailored to a particular transaction base. Transaction Library compilation and initiation, the transaction user concept, entry points to the Transaction Library, and TPS statistics are among the topics discussed.

Section 6 Update Library

This section discusses the Update Library, which is the usercoded interface between the Transaction Library and a DMSII data base. Suggested approaches and an example for coding the Update Library are included, as well as in-depth discussions of DMSII record level locking, consistency and reproducibility. Finally, extensions to the DMSII user languages for transaction processing are described.

Section 7 Remote Library And Hostinterface

This section discusses remote updating of the data base through the use of the Remote Library and the HOSTINTERFACE program.

Section 8 Exception Handling

This section describes the types of exceptions that may be returned by Transaction Library procedures. The section also discusses DMSII exception handling, including the values that can be returned from the DMSII statements OPEN, BEGINTRANSACTION, MIDTRANSACTION, and ENDTRANSACTION. In addition, error conditions and communication interruptions are explained.

Section 9 Security

This section discusses how TPS maintains security through the use of guardfiles.

Section 10 TPS Recovery

This section discusses DMSII RECOVERY and TPS recovery and how they interact. The section also discusses the synchronization of TPS with a DMSII data base as well as rollback and rebuild considerations.

Section 11 TRutility Program

This section discusses the major functions of TRUTILITY and provides syntax and semantics for each TRUTILITY statement.

Section 12 Converting To TPS

This section discusses the alternatives available to the user when converting conventional DMSII programs for an existing data base to the use of TPS. The section also discusses restrictions that may have to be imposed in order to guarantee reproducibility of results after transaction reprocessing by TPS.

Appendix A Railroad Diagrams

This section discusses how to read railroad diagrams. The user is provided with step-by-step instructions and detailed examples that clearly explain how to read railroad diagrams.

Appendix B Glossary

This section gives definitions for terms used in this manual.

Appendix C Example Of A Transaction Base

This appendix shows a sample TPS transaction base for a simple banking system. The section includes Update Library examples and user-written program examples in ALGOL, COBOL/COBOL74, and a sample ALGOL "detanking" procedure.

Appendix D SystemTR Transaction Format

This section describes the transaction format that is reserved for use by the system.

RELATED DOCUMENTS

The following A Series documents contain information that is related to the material in this manual.

Document	Form Number
ALGOL Reference Manual	1169844
CANDE Reference Manual	5014541
COBOL ANSI74 Reference Manual	1169877
COBOL Reference Manual	1169786
DMSII User Language Interface Software Operation Guide	1180536
DMSII DASDL Reference Manual	1163805
DMSII Inquiry Software Operation Guide	1164035
DMSII Database Certification Software Operation Guide	1164050
DMSII Data Dictionary Reference Manual	1177128
DMSII Utilities and Reference Guide	3001803
I/O Subsystem Reference Manual	5014483
System Software Site Management Reference Manual	1170008
System Software Support Reference Manual	1170016
System Software Utilities Reference Manual	1170024
Work Flow Language Reference Manual	1169802

2 OVERVIEW

This section introduces the most basic components of TPS, and explains how they interact. Figure 2-1 illustrates the conventional method of performing data base processing, where a program coded by the user communicates directly with the data base through the standard DMSII user language interface. TPS improves on the conventional method of data base processing by partitioning the DMSII user-written program into two logical pieces. One of these pieces is simply called the "user-written program," and the other is a module containing routines that communicate directly with the data base through the DMSII user language interface.

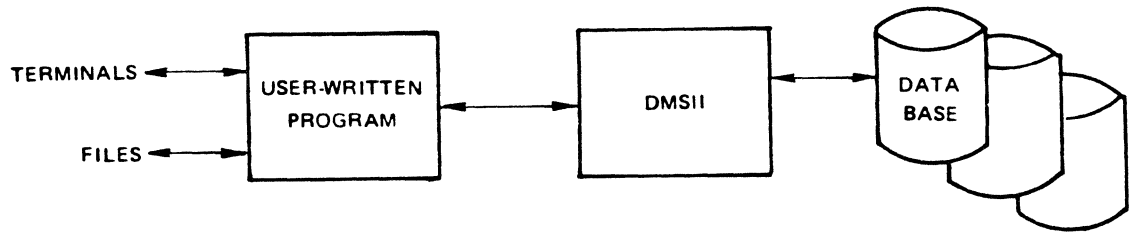


Figure 2-1. Conventional DMSII Environment

Figure 2-2 illustrates the conversion of an existing DMSII environment to a TPS environment, where the user-written program communicates only with TPS, which in turn communicates with the module that invokes the data base. (The user-written program in Figure 2-2 does not invoke the data base.) Utilizing TPS enables the user to take advantage of special functions regarding recovery, automatic reprocessing of transactions, and other features which are further explained in the paragraphs that follow.

When developing a system that uses TPS, the user must first define each update routine and inquiry routine to be performed against the database. The user must also define the data to be input to and output from each routine. A typical routine for a banking application might involve depositing a specified amount of money into a customer's account. For this routine, the following would have to be defined:

1. The input data required by the routine, such as an account number and deposit amount.

2. The output data generated by the routine, such as an updated account balance.

3. The data base manipulation required to perform the deposit.

(Refer to Appendix C for further information on a banking application.)

As shown in Figure 2-2, the user-written program collects and sends the input data to the module that invokes the data base through TPS. Based on the input data, the module that invokes the data base selects the appropriate database update or inquiry routine and performs the required action against the data base. After completing the action, the routine sends the output data back to the user-written program through TPS.

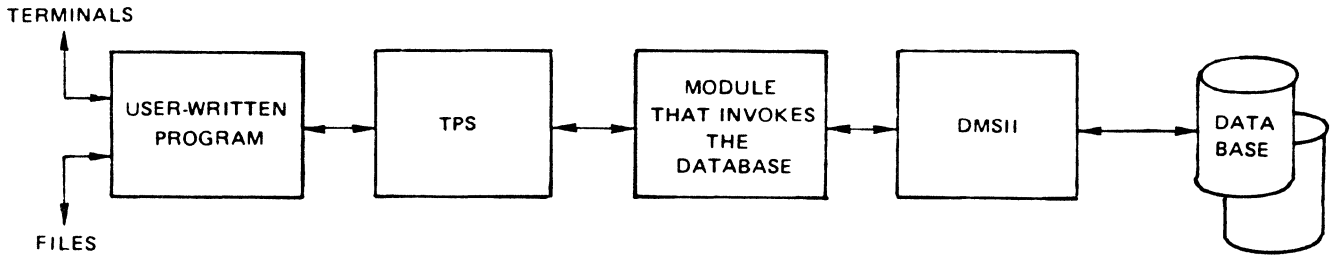


Figure 2-2. Converting to TPS environment

TRANSACTIONS AND TRANSACTION RECORDS

In TPS terms, the following three entities are collectively referred to as a "transaction":

1. The input data.

2. The routine that performs the desired processing against the data base.

3. The output data.

Usually, many transactions are defined for a single data base, but all transactions are subject to the following limitations:

1. A transaction should be a relatively small unit of processing, involving at most one action (update or inquiry, as explained in later sections of this manual) against the data base. Therefore, a single transaction should not make sweeping changes to a data base.

2. A transaction should not use any results of previous transactions, and should depend only on the state of the data base and its input data. This is the only sense in which a transaction "depends" on previous transactions.

3. Each update transaction must be programmed to ensure its independence from all other transactions. (Concepts and conventions that facilitate independence are explained section 6 of this manual.)

4. While TPS is being used to update a data base, no conventional update programs may run (though a method is provided to ease this restriction when converting existing DMSII update programs to use TPS). However, conventional inquiry programs may run in parallel with TPS. For more information on converting to the use of TPS, refer to section 12 of this manual.

Each limitation is described in greater detail in later sections of this manual.

In the TPS environment, the vehicle for sending the input data to the module that invokes the data base and receiving the output data from this module is known as a "transaction record." A transaction record is a structured variable that contains user-defined data items and system-defined control items. The user-defined data items are similar to the data items in a DMSII data set record or a COBOL 01-level variable. The TPS user defines the formats for the particular transaction records using a special Burroughs language called the Transaction Formatting Language (TFL). A format is a definition of the data items to be contained within a particular transaction record. The transaction record formats defined with TFL are processed by the TFL processor to produce a description file for the entire "transaction base." (A transaction base includes the software and files that constitute a TPS interface to a data base. Refer to section 3 for more thorough definitions of transaction base and description file.)

UPDATE LIBRARY

The user defines the routines that perform data base processing in standard DMSII user language statements, and these routines are contained in the module that invokes the data base. In the TPS environment, this module is known as the "Update Library." (When used in this manual, the term "library" refers to a particular Burroughs software construct. For more information on Burroughs libraries, refer to the A Series System Software Utilities Reference Manual.) After defining the update routines with user language statements, the user writes update programs that communicate with these routines through TPS instead of updating the data base directly.

Because the data base-knowledgeable code is isolated in the Update Library, user-written programs need only do the following:

1. Collect input data from terminals, files, or other input sources.

2. Assemble a transaction input record from the input data.

3. Access the appropriate routine in the Update Library.

4. Disassemble the transaction output record and output the result to a terminal, a file, a printer, or another appropriate medium.

TRANSACTION LIBRARY

To access an appropriate routine in the Update Library, user-written programs do not call the Update Library directly. Instead, they call procedures in the main software component of TPS, known as the "Transaction Library." The Transaction Library is a collection of procedures or entry points that accept input transaction records for processing. (The user tailors the Transaction Library to suit particular processing needs by compiling a transaction base description file with other source files, as explained in Section 5 of this manual.) The Transaction Library, in turn, calls the Update Library, which selects an appropriate routine for processing the transactions against the data base. The resulting output transaction records are returned to the user-written program by means of the Transaction Library. (See Figure 2-3 for TPS naming conventions.)

TRANSACTION JOURNALS

In addition to routing transaction records to the Update Library, the Transaction Library enables the user to collect or save transaction records in special TPS files called "transaction journals." In the TPS environment, a transaction journal is composed of a control file and one or more data files. "Tank journals" allow the user to "tank" or batch transaction records that are not processed against the data base. The TRHISTORY journal saves or audits a history of all transaction records that have been applied to the data base.

Auditing a history of all transactions that have been applied to the data base allows TPS to automatically resubmit transactions that were backed out by DMSII RECOVERY. By coding the Update Library routines in a specific way (fully explained in section 6 of this manual), the user can reproduce previous results after transactions have been resubmitted by TPS.

One important distinction between tank journals and the TRHISTORY journal relates to communication with the Update Library. Transaction records that are saved in the TRHISTORY journal are sent from the user-written program to the Transaction Library, and from there to the Update Library for processing. After processing, output transaction records are sent back to the Transaction Library, and from there back to the user-written program. However, transaction records that are stored in tank journals are never sent to the Update Library; no processing of tanked transactions occurs.

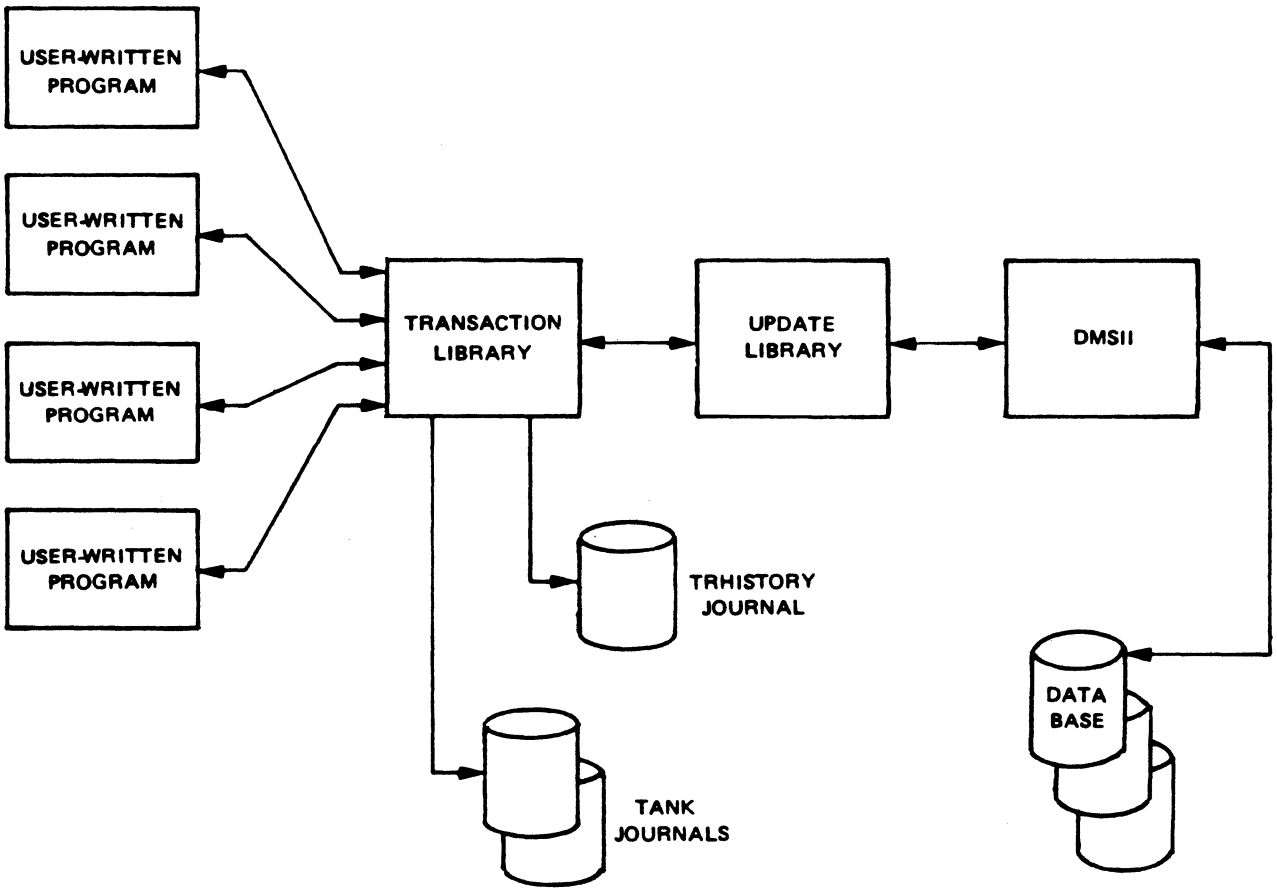


Figure 2-3. TPS Naming Conventions

REMOTE LIBRARY

Another TPS feature is its distributed processing capability, illustrated by Figure 2-4. The TPS user can process transactions from both a host system (the system where the data base resides) and any number of remote systems, which are connected through a Burroughs Network Architecture (BNA) data communications network, without having to change user-written programs. The portability of TPS user-written programs results from the fact that they always call the same entry points to process transactions, regardless of where the programs are executed. Except for speed, the fact that the data base might be on another system is transparent to the user-written program.

A special module called the "Remote Library" (shown in Figure 2-4) is used to enable remote-host communications. The Remote Library is compiled on the host system using the transaction base description file. It must then be copied over to the remote system. Thus, a program on the remote system calls the same entry points that a program on the host system calls, except that the program on the remote system is actually calling the Remote Library.

HOSTINTERFACE PROGRAM

The Remote Library does not directly call the Update Library on the host system in the same way that the Transaction Library does. Instead, the Remote Library sends input transaction records to the "HOSTINTERFACE program," a TPS module that resides on the host system. (A BNA "port file" interface is used to send transaction records from the remote system to the host system.) The HOSTINTERFACE program substitutes for the remote user-written program on the host system, and its job is to call the appropriate entry point in the Transaction Library for the input transaction record that was sent. After processing, the HOSTINTERFACE program sends the output transaction record back to the Remote Library which, in turn, returns the transaction record to the remote user-written program.

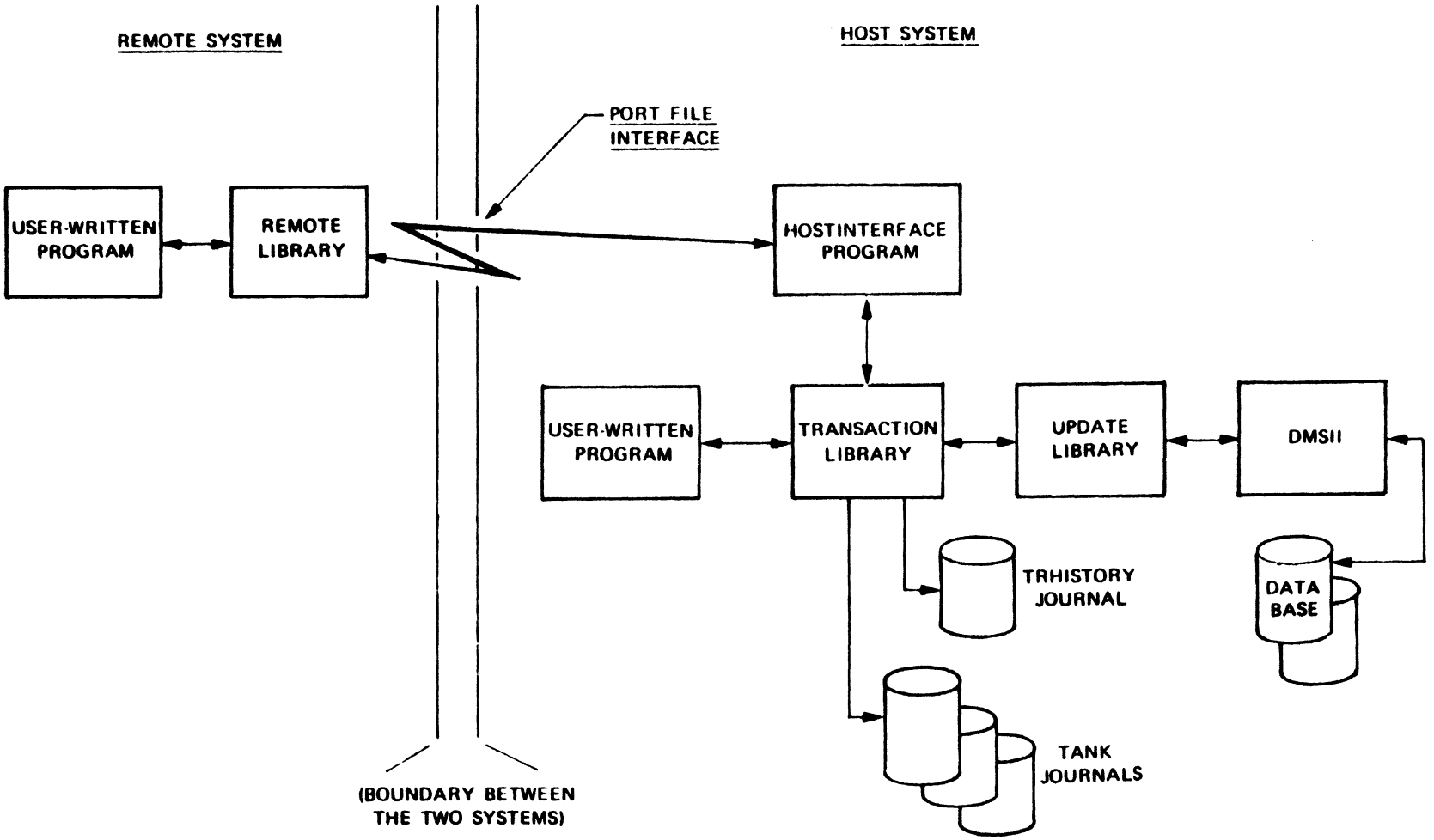


Figure 2-4. Remote Real-Time Transaction Processing with TPS

TPS SOFTWARE COMPONENTS

The following software components are provided to the TPS user:

SYMBOL/TFL

SYMBOL/TFL is the source file for the TFL processor, named SYSTEM/TFL. The user defines transaction record formats using TFL and creates a transaction base description file by running SYSTEM/TFL (described in the following paragraph). The data submitted to SYSTEM/TFL is usually referred to as the "TFL source," and the transaction base description file is usually referred to as the "TRDESCRIPTION" file.

SYSTEM/TFL

SYSTEM/TFL is a program that functions as a processor. SYSTEM/TFL reads the symbolic description of transaction record formats written in TFL and produces a TRDESCRIPTION file. Depending on TFL processor options specified by the user, SYSTEM/TFL can perform other functions, as well. For example:

1. SYSTEM/TFL can produce a printer listing of the TFL source file, including errors that have been detected.

2. If UPDATE is specified in the TFL source, SYSTEM/TFL can create an updated TRDESCRIPTION file.

3. If the TFL ZIP option is specified, SYSTEM/TFL can initiate compilation of all TPS tailored software.

SYSTEM/TRINTERFACE

SYSTEM/TRINTERFACE is an external coroutine that communicates with the user language compilers to invoke and compile references to the user language interfaces for TPS. These references include user-defined transaction record formats, transaction journals, subbases, and other information related to transaction processing.

TRBASE/HOSTINTERFACE

TRBASE/HOSTINTERFACE is a source file that is compiled to create an interface between the host system in a network and a user-written program running on a remote system. One such interface program runs for each user-written program on a remote system. When a user-written program attempts to call a Transaction Library entry

point on the remote system, the interface program actually makes the entry point call for the user-written program on the host system.

TRBASE/HOSTLIB

TRBASE/HOSTLIB is a source file compiled to produce the Transaction Library. The Transaction Library is a collection of procedures or entry points called by user-written programs to process or tank transactions and read them back from transaction journals.

TRBASE/PROPERTIES

TRBASE/PROPERTIES is a compiler file that is made up of ALGOL defines and procedures and is compiled with other TPS software to provide information about the format of the TRDESCRIPTION file and the system format. For example, TRBASE/PROPERTIES is compiled with the TRDESCRIPTION file and the source file TRBASE/HOSTLIB to produce a user-tailored Transaction Library.

TRBASE/REMOTELIB

TRBASE/REMOTELIB is a source file that is used in a distributed processing environment. It is compiled on the host system and then copied over to the remote system, where it is known as the Remote Library. It runs with the user-written programs, and is analogous to the Transaction Library that runs on the host system, but does not perform all of the functions of the Transaction Library. The Remote Library has the same entry points as the Transaction Library and appears identical to the user.

TRBASE/UTILITY

The TRBASE/UTILITY program is tailored by the user to meet specific transaction processing needs, and has four major functions:

1. Initializing transaction journal control files.
2. Searching and printing portions of transaction records in transaction journal data files.
3. Printing information in transaction journal control files.
4. Recovering transaction journal control files.

3 TRANSACTION FORMATTING LANGUAGE (TFL)

Transaction Formatting Language (TFL) is a symbolic language employed by the user to define transaction record formats, subformats, journals, subbases, and other information related to transaction processing. TFL is similar in several respects to DASDL (Data and Structure Definition Language), which is used, among other things, to define the formats of data set records. The symbolic description of transaction record formats, journals, and other related structures is collectively referred to as a transaction base. A transaction base is the analog of a data base, and can be defined for at most one data base.

The TFL source file describing a transaction base is processed with the TFL Processor, which is called SYSTEM/TFL, as shown in Figure 3-1. (SYSTEM/TFL is described later in this section.) SYSTEM/TFL produces a transaction base description file, referred to as the TRDESCRIPTION file. The TRDESCRIPTION file is used by the user language compilers when compiling user-written programs which invoke a transaction base and when compiling tailored TPS software.

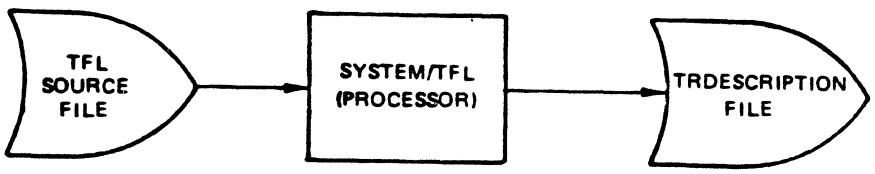


Figure 3-1. Generating the Description of a Transaction Base

TFL BASIC CONSTRUCTS

The following paragraphs give brief descriptions of the most basic TFL constructs that commonly appear in TFL descriptive statements.

Character Set

TFL uses the following EBCDIC characters:

0-9			Space or blank
A-Z	Uppercase only	,	Comma
-	Hyphen or minus	;	Semicolon
+	Plus sign	:	Colon
*	Asterisk	.	Period or decimal point
/	Slash	"	Quotation mark
=	Equal sign	$	Dollar sign
>	Greater than symbol	%	Percent sign
<	Less than symbol	(	Left parenthesis
		)	Right parenthesis

Although TFL uses only the characters in the preceding list, <string>s can contain any EBCDIC character.

Digit

Letter

Syntax:

<letter>

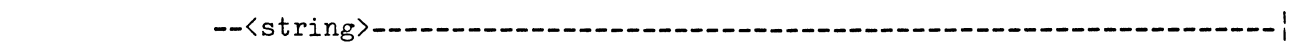
-- one of the EBCDIC characters, A thru Z

Comment

<comment>s are TFL descriptive information about a transaction base, which are stored in the TRDESCRIPTION file. When a COBOL program invokes a transaction base, the <comment>s are listed with each <transaction record format> and <transaction item>, if the LIST dollar-card option is set in the program. For ALGOL programs, <comment>s are listed with <transaction record format>s and <transaction item>s when the LISTDB dollar option is set in the program.

Syntax:

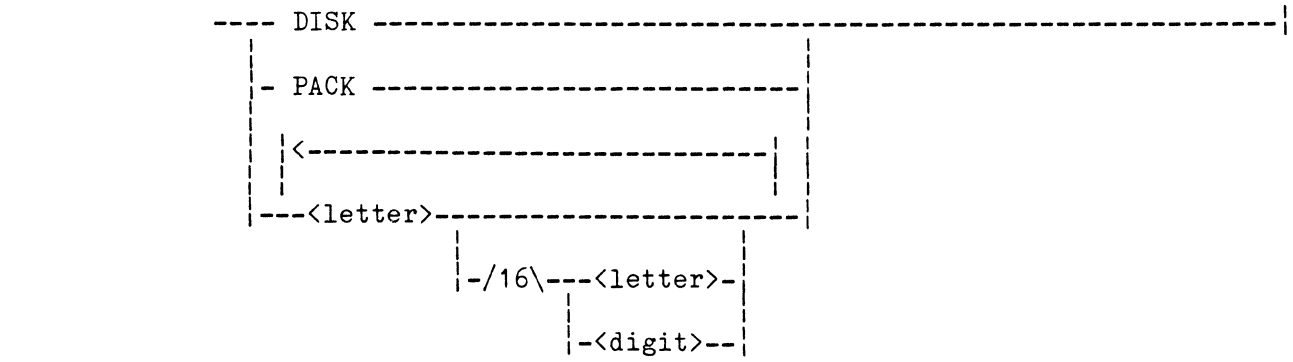
<comment>



Family Name

A <family name> identifies a disk or disk pack family.

Syntax:



Simple Identifier

A <simple identifier> consists of uppercase alphanumeric characters and single embedded hyphens. The first character must be an alphabetic character. A simple identifier may be at most 17 characters in length.

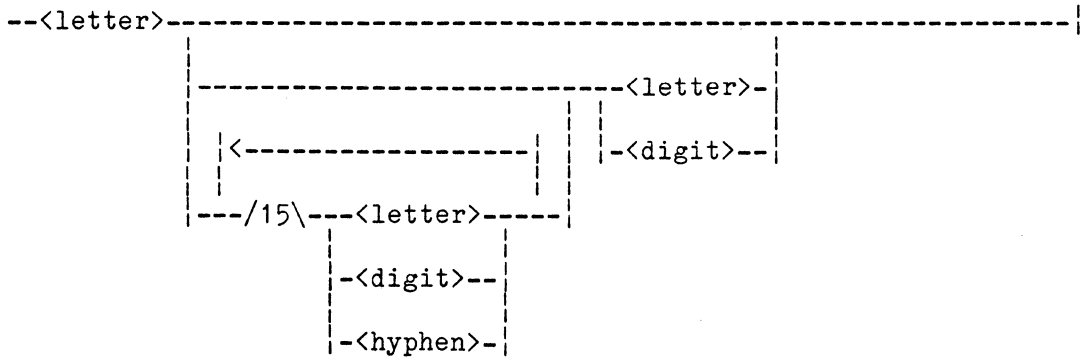
Legal Identifiers:	ABC
	ABC-EF-HIJKLMNOPQ
	STEP-1
Illegal Identifiers:	3DE
	abc
	ABCDEFGHIJKLMNOPQR
	-ABC
	ABC-

The BDMSCOBOL and BDMSCOBOL74 compilers support the hyphenated identifier capability. Identifiers should be referenced as they are declared in the TFL source. For the BDMSALGOL language, hyphenated identifiers that appear in the TFL source must be referenced as

underscored identifiers. For example, the TFL source identifier called CREATE-ACCOUNT must be referenced as CREATE_ACCOUNT in all BDMSALGOL programs.

Syntax:

<simple identifier>



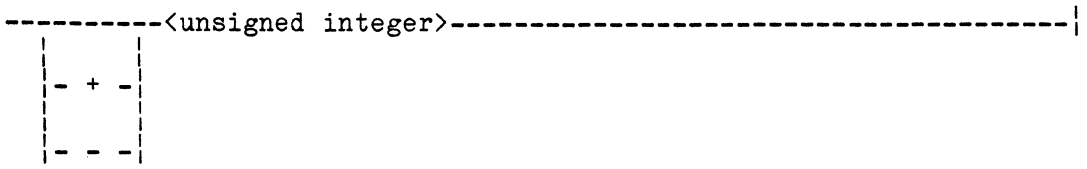
Integer

An <integer> is used to represent a signed or unsigned whole value. A "+" or "-" immediately followed by an <unsigned integer> with no intervening space is always taken as a sign, never an operator.

An <unsigned integer> represents an unsigned whole value.

Syntax:

<integer>



<unsigned integer>

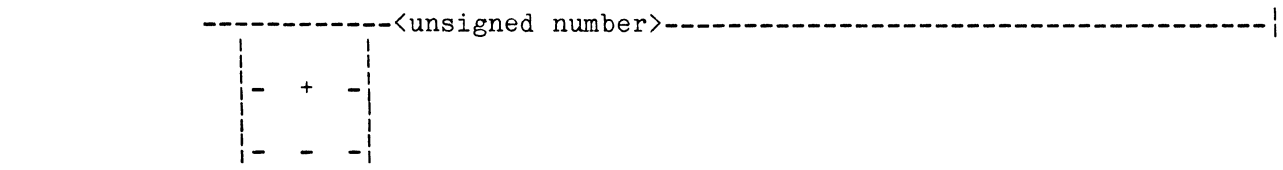
Number

A <number> is used to represent a signed or unsigned fractional or whole value. A + or - immediately followed by an <unsigned number> with no intervening space is always taken as a sign, never an operator.

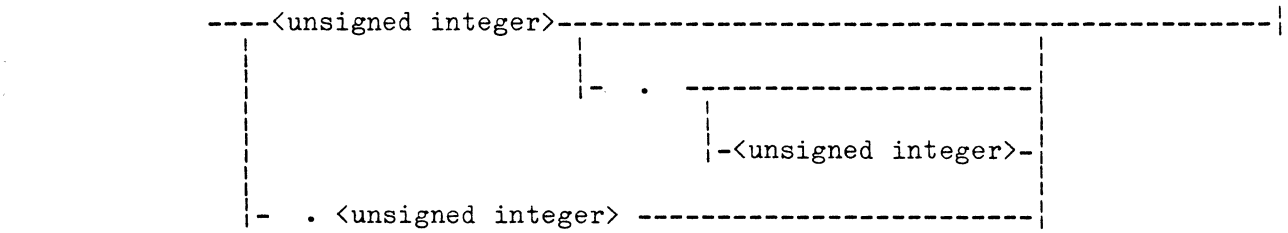
An <unsigned number> represents an unsigned fractional or whole value.

Syntax:

<number>



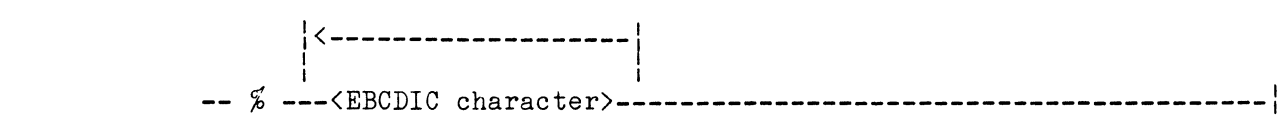
<unsigned number>



Remark

<remark>s are preceded by a percent sign and can appear anywhere, except within <string>s. When a <remark> is encountered, the remainder of the input record is ignored and processing of the next record begins.

Syntax:



String

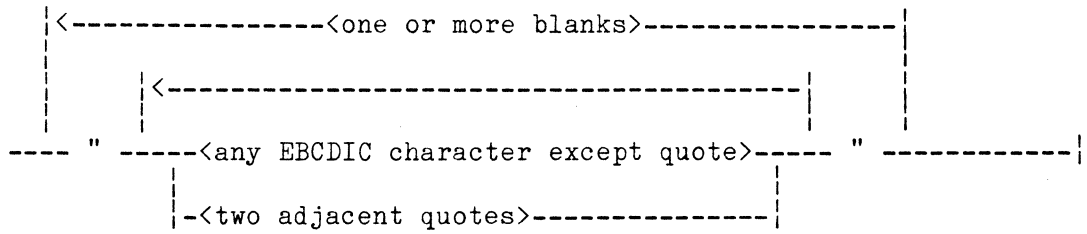
<string>s must be contained within quotes. If a <string> extends beyond column 72 of a record, the record boundary acts as an implicit quote. The <string> must be continued on the next record and must begin with a quote.

A quote is represented in a <string> by placing two adjacent quotes on the same record.

Two adjacent <string>s separated only by blanks, <remark>s, or record boundaries are automatically concatenated.

A <string> can contain at most 255 characters.

Syntax:



Expression

<expression>

Arithmetic Expression

<arithmetic expression>s yield numeric values. They are formed by combining <data item>s and constants using arithmetic operators.

The following arithmetic operators are provided:

+	Add
-	Subtract
*	Multiply
/	Divide
DIV	Integer Divide
MOD	Remainder Divide

The sequence in which operations are performed is determined by the precedence of the operators involved. The order of precedence is as follows:

First:	*, /, DIV, MOD
Second:	+, -

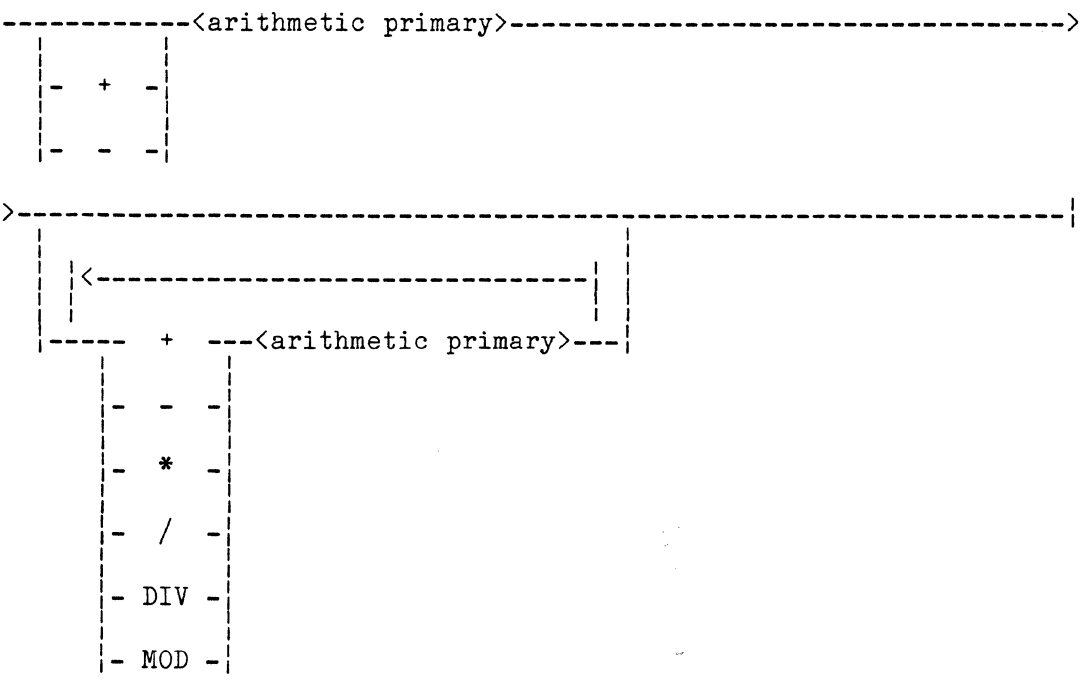
When operators have the same precedence, the operations are performed in order from left to right. Parentheses can be used in standard mathematical fashion to override the usual order of evaluation.

When occurring <data item>s are referenced, all subscripts must be specified. Subscripts must be <unsigned integer> constants.

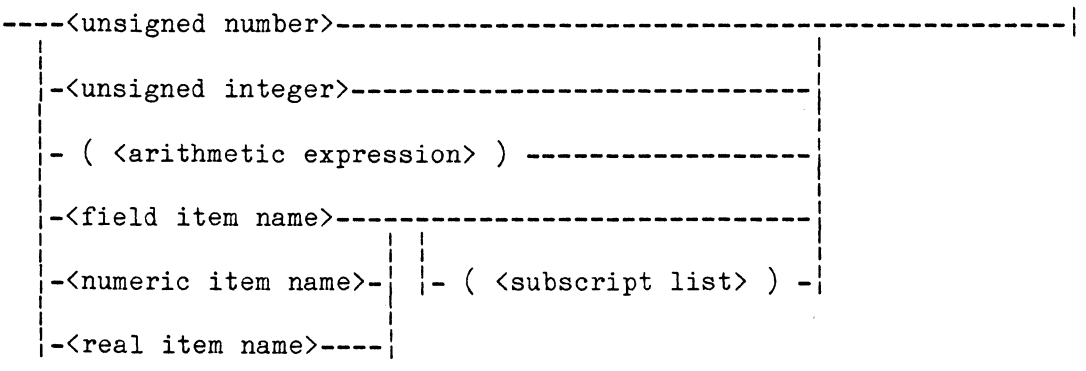
Negative numbers must be enclosed within parentheses when they follow operators.

Syntax:

<arithmetic expression>



<arithmetic primary>



<subscript list>

Boolean Expression

<Boolean expression>s yield a value of either TRUE or FALSE. They are formed by combining <data item>s, <arithmetic expression>s, and <string>s using relational and logical operators. The following rules of precedence control the sequence in which <Boolean expression>s are evaluated.

First:	<arithmetic expression>
Second:	<arithmetic compare>,<string compare>
Third:	NOT
Fourth:	AND
Fifth:	OR

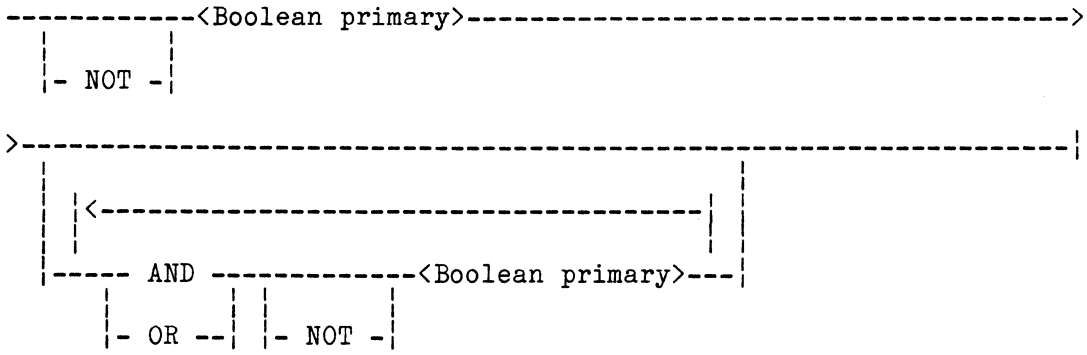
When operators have the same precedence, the operations are performed in order from left to right. Parentheses can be used to override the usual order of evaluation.

Occurring <data item>s must be fully subscripted. Subscripts must be <unsigned integer> constants.

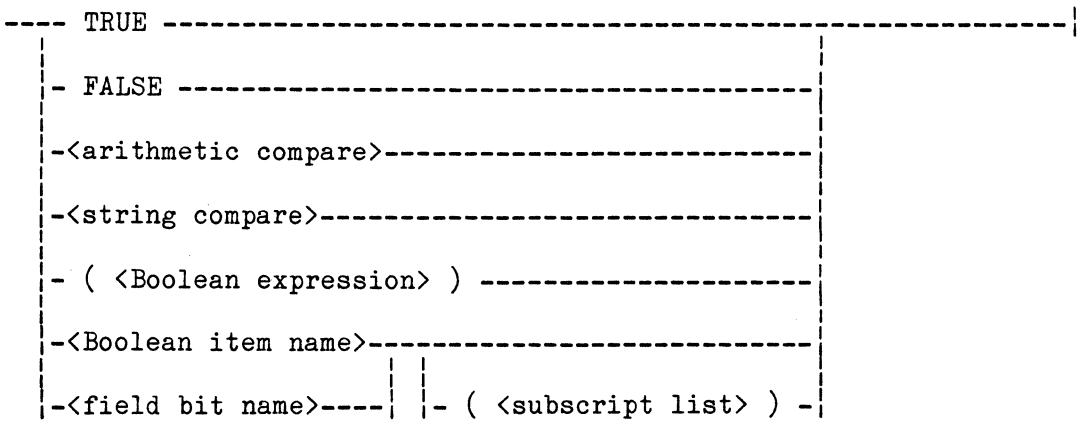
When comparing <alpha item>s, the comparison is based on the number of characters contained in the <alpha item> or <string> having the shorter length.

Syntax:

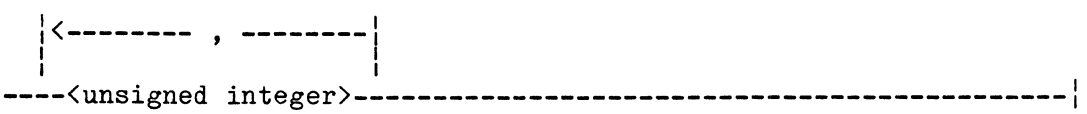
<Boolean expression>



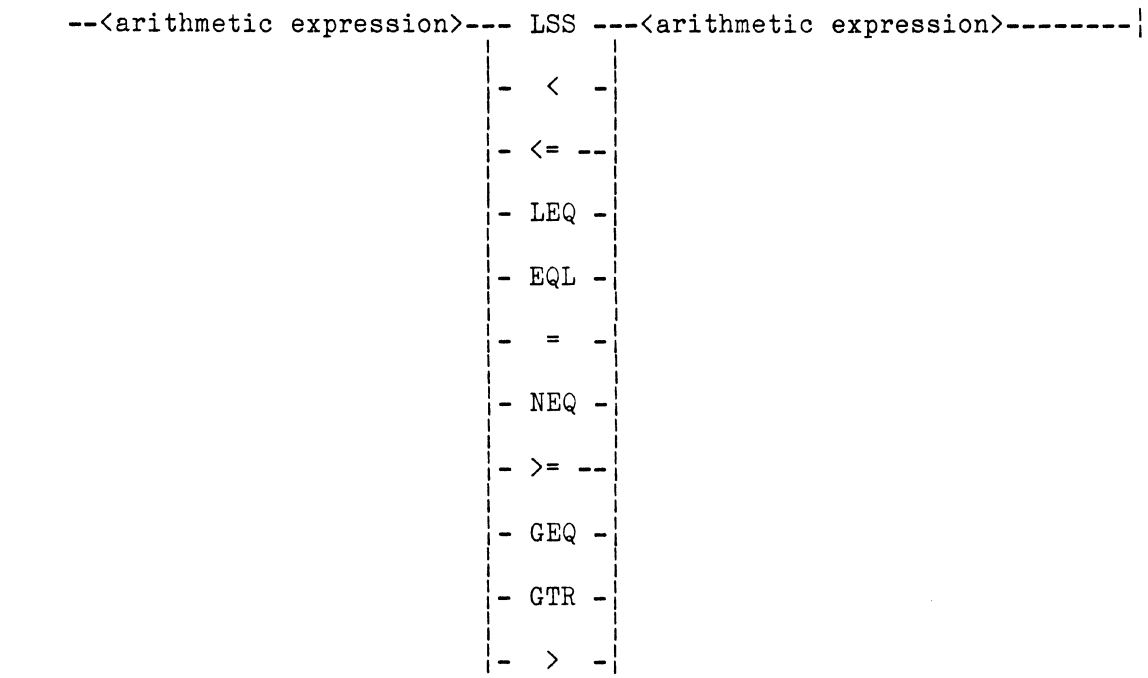
<Boolean primary>



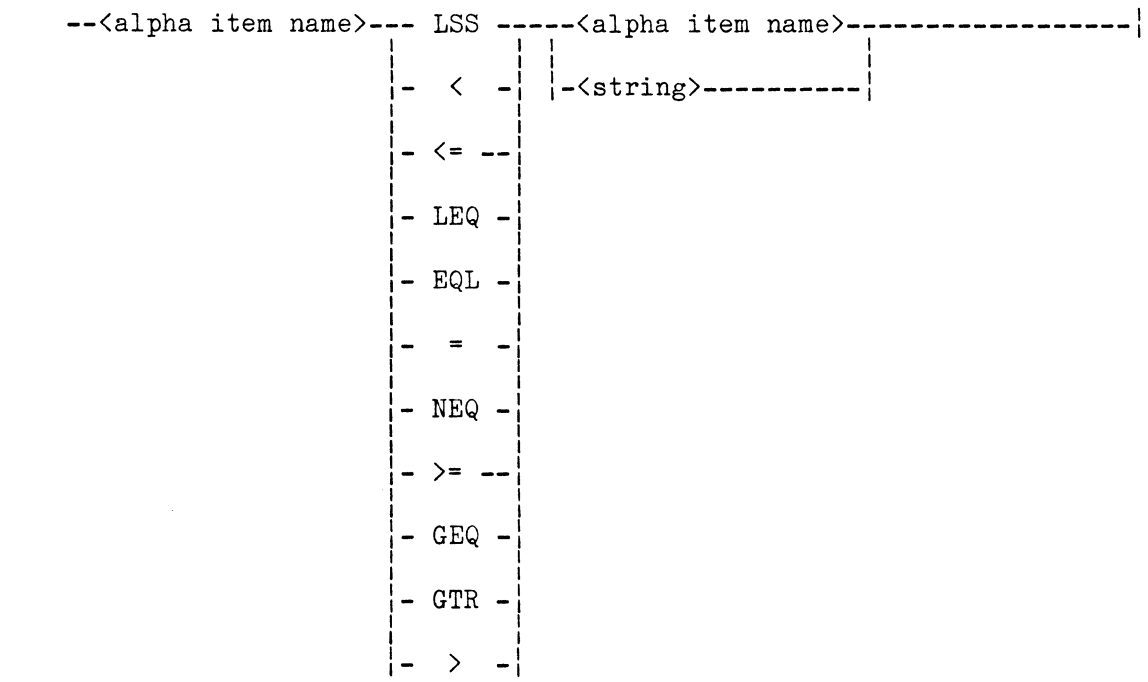
<subscript list>



<arithmetic compare>



<string compare>

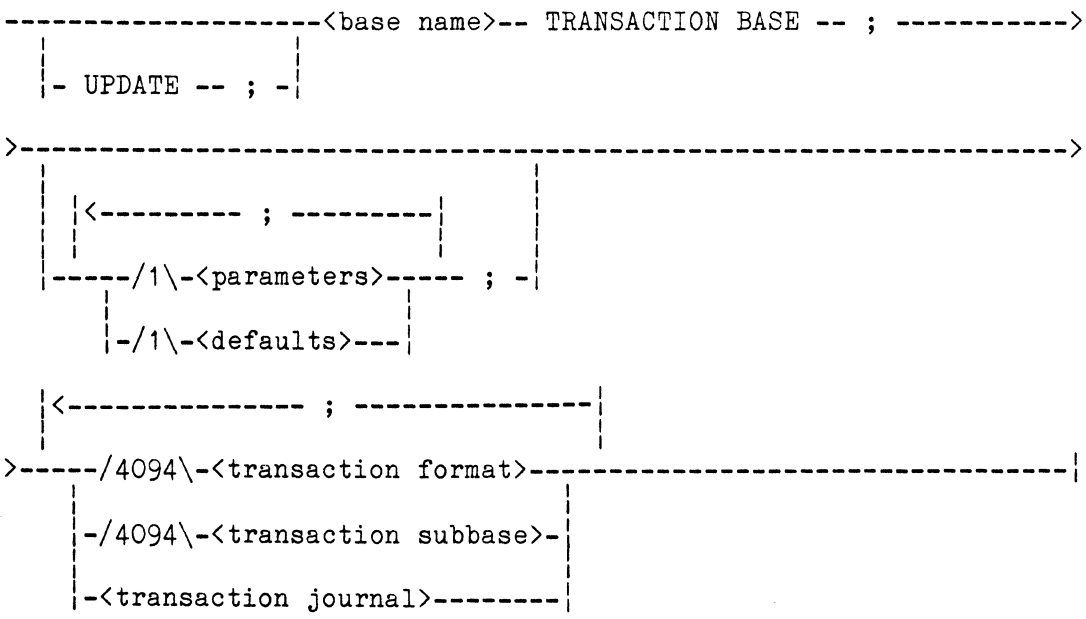


TFL DESCRIPTION OF TRANSACTION BASE

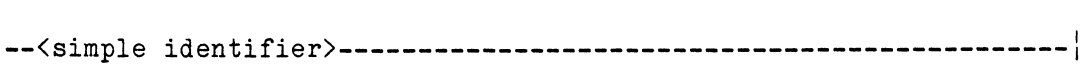
TFL is used to define a transaction base. A transaction base consists of several components, which are described later in this section.

Syntax:

<transaction base>



<base name>



Semantics:

<base name> TRANSACTION BASE

Assigns <base name> as the name of the new transaction base. The file that is created using this description is named <base name>/TRDESCRIPTION.

UPDATE

Causes a previous TRDESCRIPTION file to be updated with the correct input. When an update is performed, the entire transaction base must be redescribed. TFL processes the new source file and then compares it to the old source file to ensure that all changes are valid. The <update option> is further discussed under "Updating Transaction Bases" in this section.

<parameters>

Defines values for the global transaction base properties. For more information, refer to "Parameters" in this section.

<defaults>

Assigns default values to control file specifications, data file specifications, and items. For more information refer to Defaults in this section.

<transaction record format>

Defines the format that a single transaction can assume. A <transaction record format> defines the <data item>s and <group item>s that can be contained within a particular transaction record. There may be at most 4095 <transaction record format>s in a <transaction base>. For more information refer to "Transaction Record Format" in this section.

<transaction subbase>

A list of <transaction record format>s and <transaction subformat>s that are invoked together by a user-written program. There may be at most 4095 <transaction subbase>s in a <transaction base>. A <transaction subbase> is similar in many respects to a DASDL logical data base.

<transaction journal>

Defines the values of the <control file spec>s and <data file spec>s for a transaction journal. For more information, refer to "Transaction Journal" in this section.

<simple identifier>

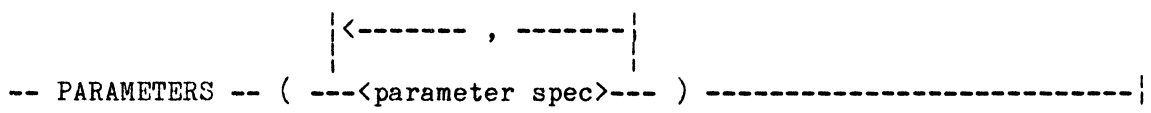
Defined under "TFL Basic Constructs" in this section.

Parameters

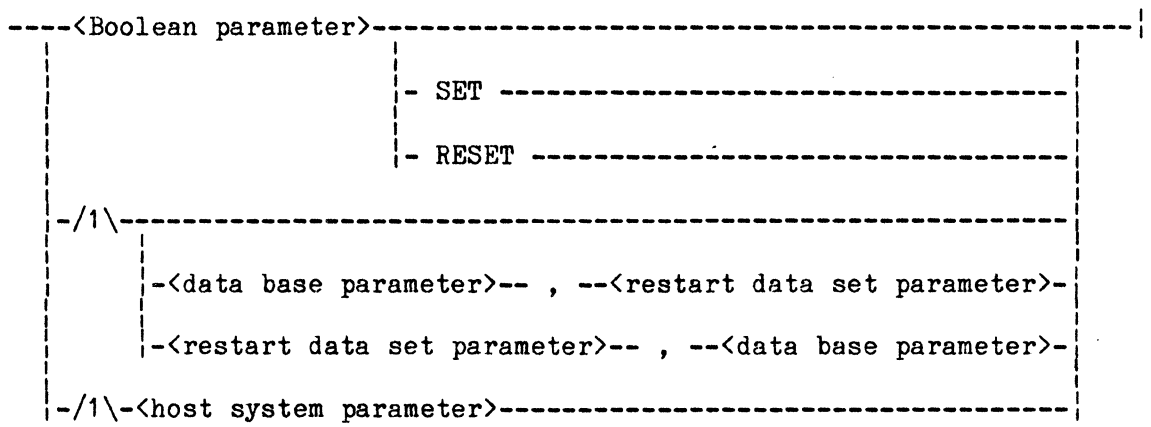
<parameters> define values for the global transaction base properties.

Syntax:

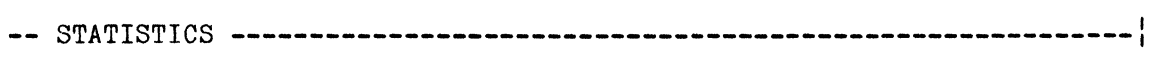
<parameters>



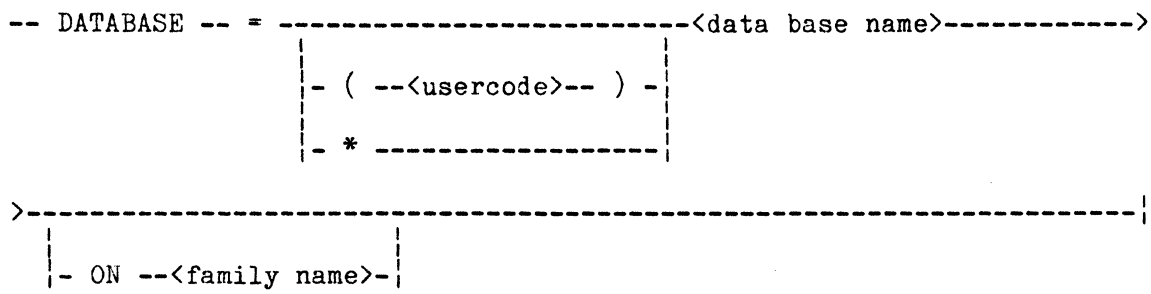
<parameter spec>



<Boolean parameter>



<data base parameter>



<data base name>

--<simple identifier>----------

<restart data set parameter>

-- RESTARTDATASET -- = --<restart data set name>----------

<restart data set name>

--<simple identifier>----------

<host system parameter>

-- HOSTSYSTEM -- = --<host system name>----------

Semantics:

<Boolean parameter>

A parameter that has a value of TRUE (if SET is specified) or FALSE (if RESET is specified). If neither SET nor RESET is specified, SET is assumed. The following <Boolean parameter> can be used:

STATISTICS

When set, STATISTICS causes the Transaction Library to gather statistics concerning the operation of the system. A report summarizing these statistics is produced on a printer file when the last user calls the Transaction Library entry point CLOSETRBASE. (For more information regarding STATISTICS, refer to Section 5 of this manual.)

<data base parameter>, <restart data set parameter>
<restart data set parameter>, <data base parameter>

If the transaction base is to access a data base, it must have access to the control file and restart data set of that data base. <data base parameter> gives the name of the data base to be accessed and specifies where to find its control file. If no <data base parameter> is specified, data base-related code is omitted from the Transaction Library. Only audited databases can be used in conjunction with TPS.

<restart data set parameter> names the restart data set within the data base, which is used by DMSII Recovery during the Halt/Load recovery process.

<host system parameter>

Specifies the name of the system with which a remote system communicates. This parameter must be included if a remote version of the Transaction Library (in other words, the Remote Library) is to be compiled. By definition, the system that contains the data base is the "host system"; all other systems are referred to as "remote systems." For information on the TPS Remote Library, refer to Section 7 of this manual.

DATABASE = <data base name>

Associates the transaction base with the specified data base.

(<usercode>)
*

<usercode> specifies the usercode where the data base control file resides. An asterisk (*) indicates that the control file resides on the system node. If neither <usercode> nor * is given, no usercode is assumed.

ON <family name>

Specifies the family (disk or pack) on which the data base control file resides. If this option does not appear, DISK is assumed.

RESTARTDATASET = <restart data set name>

Specifies the name of the restart data set declared in the DASDL for the data base named <data base name>.

HOSTSYSTEM = <host system name>

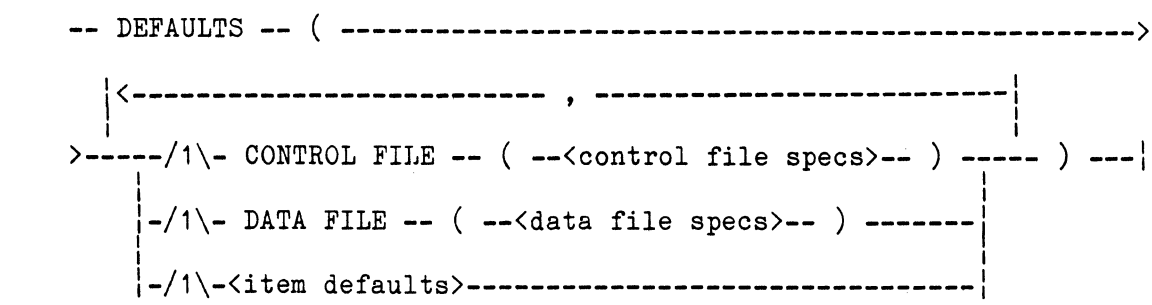
Specifies the name of the system with which all remote systems communicate.

Defaults

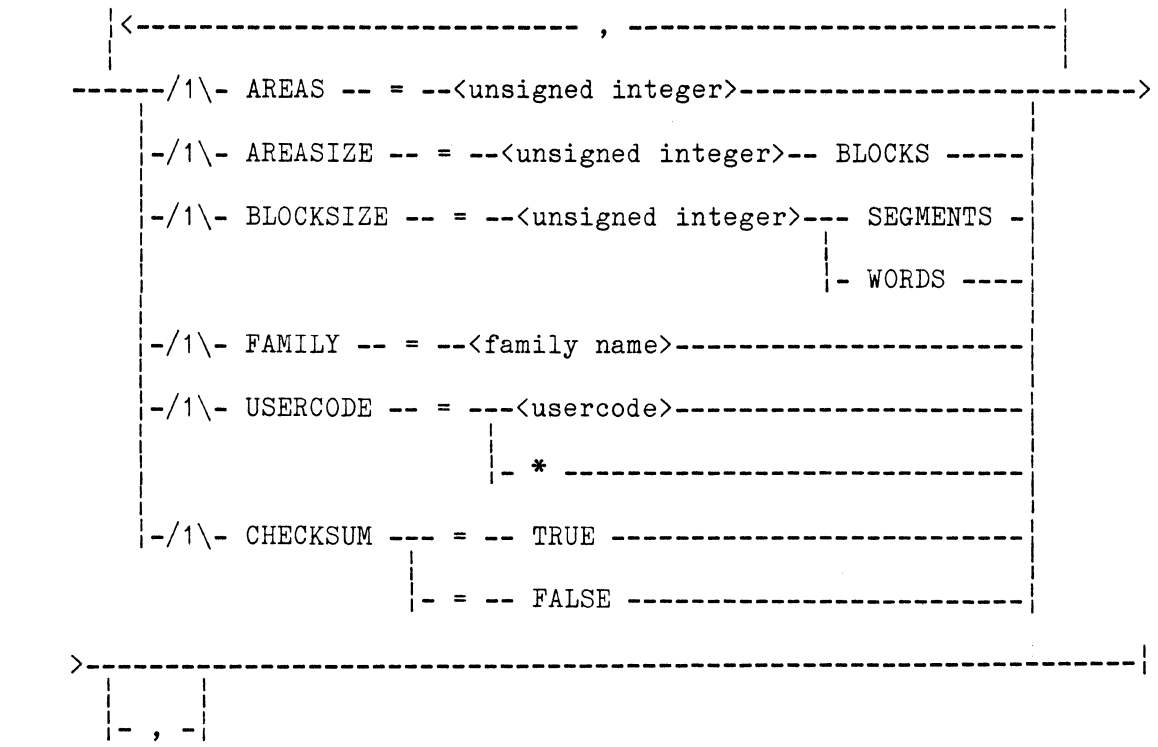
<defaults> assign default values for control file specifications, data file specifications, and items. Specifying the value of an attribute using <defaults> is equivalent to specifying the value for each item or journal file attribute in the transaction base for which the value is appropriate. The specifications in the <defaults> apply to every element of the transaction base unless they are explicitly overridden for a particular element.

Syntax:

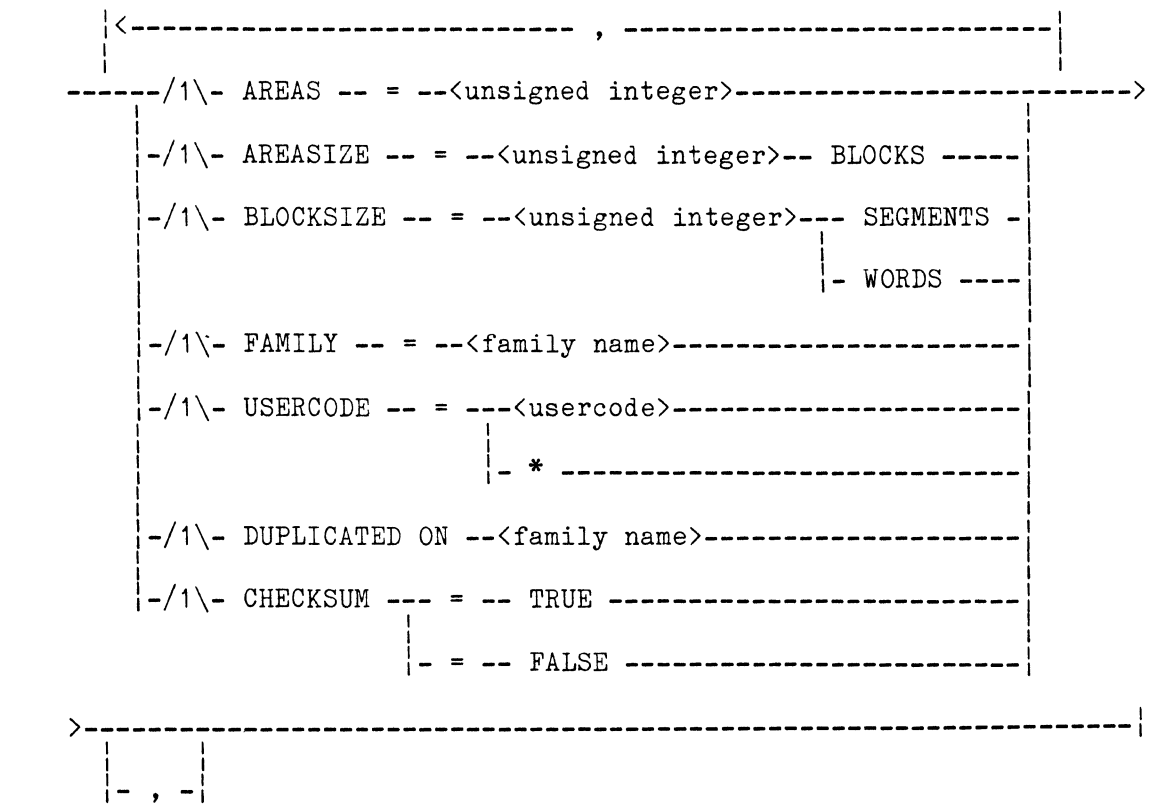
<defaults>



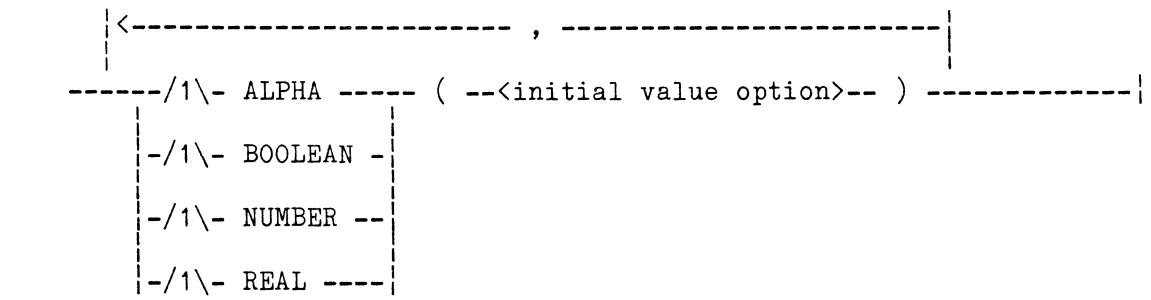
<control file specs>



<data file specs>



<item defaults>



Semantics:

<control file specs>

Assigns particular default values to file attributes and other specifications of journal control files. These default values are used instead of system default values whenever individual specifications are not provided in the TFL source. The system default values are listed under "Transaction Journal" in this section.

<data file specs>

Assigns particular default values to file attributes and other specifications of journal data files. These default values are used instead of system default values whenever individual specifications are not provided in the TFL source. The system default values are listed under "Transaction Journal" in this section.

<item defaults>

Specifies default <initial value option>s to be used for items of any of the indicated data types. These <initial value>s are used instead of system default values whenever individual initial value specifications are not provided in the description of <data item>s contained within a <transaction record format>. The system default values are listed under "Data Items" in this section.

<initial value option>

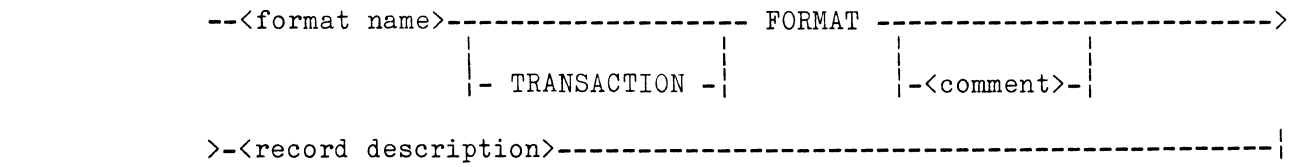
Used instead of a system default value for an item whenever an individual initial value specification is not provided in the description of a <data item> contained within a <transaction record format>. For more information, refer to INITIALVALUE under DATA ITEMS in this section.

Transaction Record Format

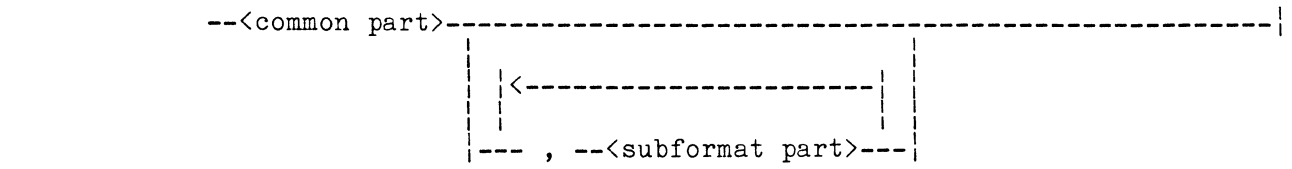
A <transaction record format> defines the format or data layout a transaction record can assume. For information on the system transaction format called SYSTEMTR, refer to Appendix D in this manual.

Syntax:

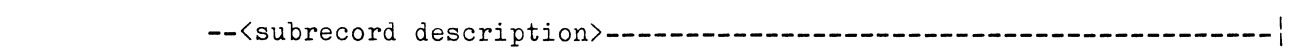
<transaction record format>



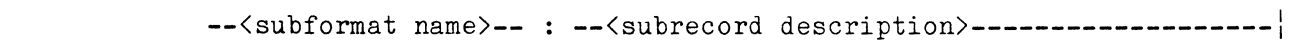
<record description>



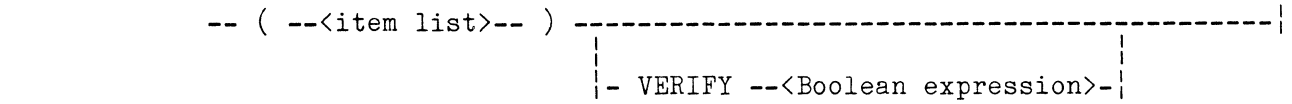
<common part>



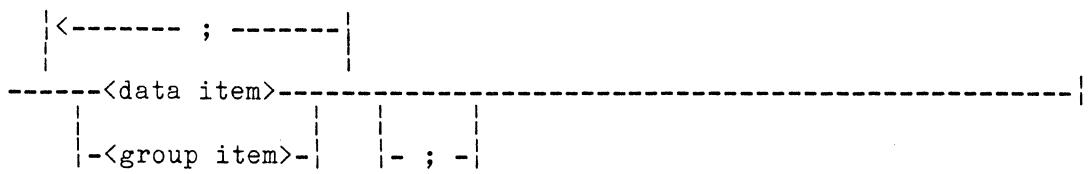
<subformat part>



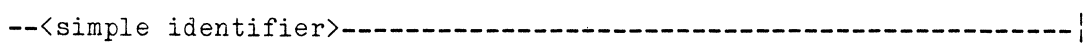
<subrecord description>



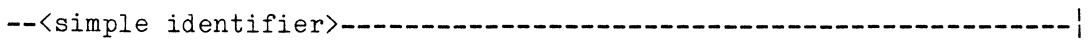
<item list>



<format name>



<subformat name>



Semantics:

<format name> FORMAT
<format name> TRANSACTION FORMAT

Identifies the description of a particular transaction format. Each <format name> is assigned a unique integer value, which is stored in the control portion of the transaction record (the TRFORMAT control item, which is defined in Section 4 of this manual). The <format name> can be used in certain contexts in a user-written program to refer to this value.

<record description>

Defines the <data item>s and associated specifications for a particular <transaction format>.

<common part>

Defines the portion of a transaction record that contains <data item>s common to all transaction records of a given format.

<subformat part>, <subformat part> ...

Defines the variable portion of a transaction record. Transaction records CREATEd in a specific transaction format and subformat contain the <data item>s common to the format, as well as the data items defined within the specified subformat.

Data items with the same name that are contained within different subformats of the same format must be declared with the same type and size. Such items must also occupy the same position within their respective subformats.

<subformat name>

Identifies the description of a particular transaction subformat. Each <subformat name> is assigned an integer value, which is stored in the control part of the transaction record (the TRSUBFORMAT control item, which is defined in section 4 of this manual). The <subformat name> can be used in certain contexts in a user-written program to refer to this value. Though <transaction record format> names must be unique, transaction <subformat name>s need not be unique. That is, two different <transaction record format>s can contain transaction subformats having the same name.

<subrecord description>

A list of item descriptions, optionally followed by a VERIFY condition. For any transaction record, the applicable VERIFY condition(s) must yield the value TRUE when evaluated using the items of that record. A <Boolean expression> is used to ensure that only valid transaction records are accepted by the Transaction Library. Each time a user program calls a Transaction Library entry point to either process or tank a transaction record, the <Boolean expression> is applied to the record. If the <Boolean expression> is not satisfied (in other words, the value returned is FALSE), the record is not accepted and the user program is notified of the error.

A <Boolean expression> can be specified for the <common part> of a transaction format, and separate <Boolean expression>s can be specified for each subformat of a transaction format. The <Boolean expression> for the <common part> can only refer to items in the <common part>. The <Boolean expression> for a subformat can refer to items in both the <common part> and the subformat to which the condition is attached. If separate <Boolean expression>s are specified for the <common part> and a subformat, then both must be satisfied before the record is stored.

<data item>
<group item>

Defined under "Data Item" and "Group Item" in this section.

<simple identifier>
<Boolean expression>

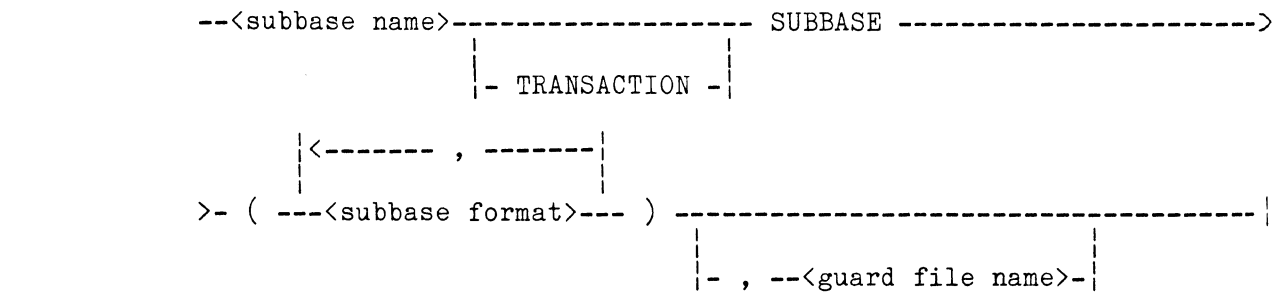
Defined under "TFL Basic Constructs" in this section.

Transaction Subbase

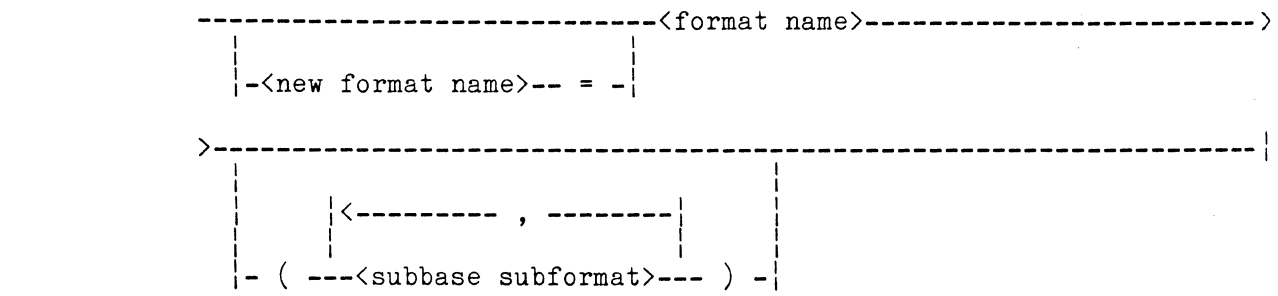
A <transaction subbase> lists transaction formats that are invoked together by a user-written program. Such formats are usually related in some way. For example, each subbase may correspond to a logical data base, so that a program invoking a subbase only opens a specific portion of the data base.

Syntax:

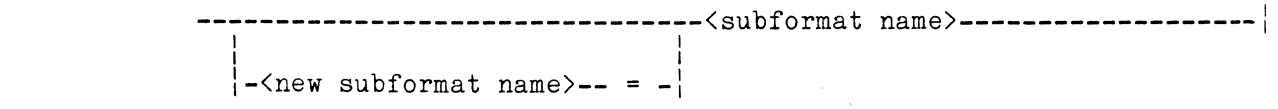
<transaction subbase>



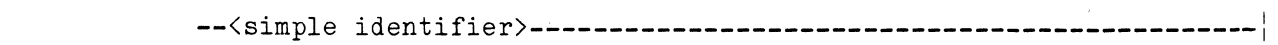
<subbase format>



<subbase subformat>



<subbase name>



<new format name>

--<simple identifier>----------¦

<new subformat name>

--<simple identifier>----------¦

Semantics:

<subbase name> SUBBASE
<subbase name> TRANSACTION SUBBASE

Each of these constructs identifies the description of a particular <transaction subbase>. Each <subbase name> is assigned a unique integer value, which is stored in the control portion of the transaction record (the TRSUBBASE control item, which is described in Section 4 of this manual).

<format name>

Identifies a <transaction record format> that is invoked as part of a particular <transaction subbase>.

<new format name> = <format name>

Gives an internal name to a <transaction record format>. User-written programs then refer to the <new format name> rather than the <format name>.

(<subbase subformat>, <subbase subformat>, ...)

Restricts the invoked subformats to those listed. If this list does not appear, all subformats are invoked.

<guardfile name>

Associates the specified guardfile with a subbase. The guardfile restricts the use of particular <transaction record format>s and <transaction subformat>s to a specific set of usercodes or programs, or both. In other words, unauthorized usercodes and programs are not allowed to process transactions contained within a subbase that is protected by a guardfile. By checking with the guardfile, the system verifies that the usercode and program name are authorized to access the subbase. For more details on TPS security, refer to Section 9.

<subbase subformat>
<subbase subformat> = <new subformat name>

Identifies one of several subformats to be invoked along with a <transaction subbase> format.

The <new subformat name>, if present, gives an internal name to a <transaction subformat>. User-written programs then refer to the <new subformat name> rather than the <subformat name>.

Transaction Journal

There are two types of transaction journals: a history journal and one or more tank journals. A tank journal contains transaction records that have been accumulated during a tanking operation but not yet processed (applied to the data base). A transaction base can have several tank journals with any name of the user's choice except TRHISTORY.

A history journal contains transaction records that have been processed against the data base. The transaction records are written in the order in which they must be reprocessed during a recovery operation. A transaction base can have only one history journal, and its journal name must be TRHISTORY.

Each transaction journal contains the following physical files:

1. Transaction data files

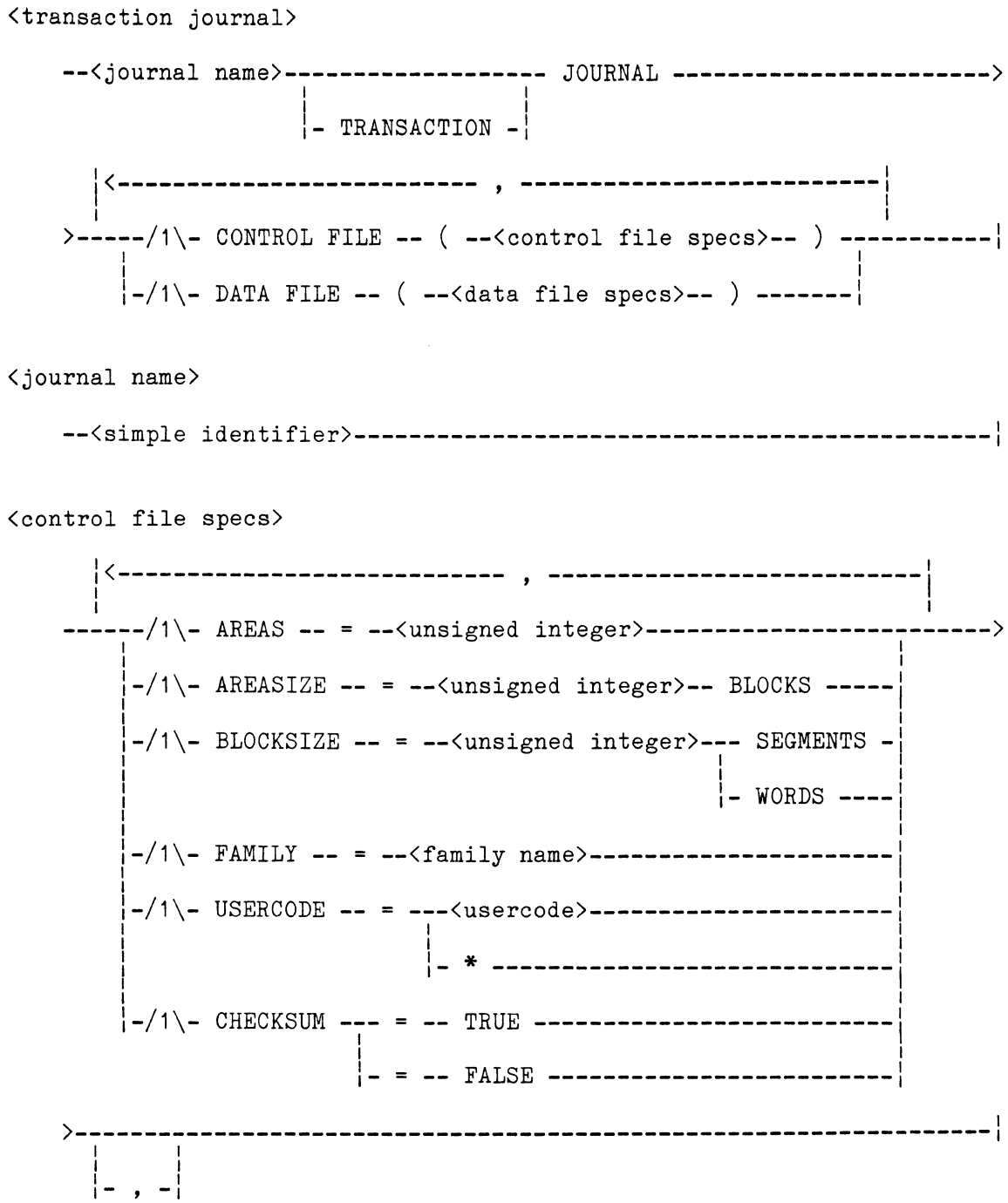
   These files contain variable-length transaction records in variable-length blocks. Each file has a title of the form <base name>/<journal name>/<file number>, where <file number> ranges from 1 through 9999, wrapping around from 9999 to 1.

2. Transaction control file

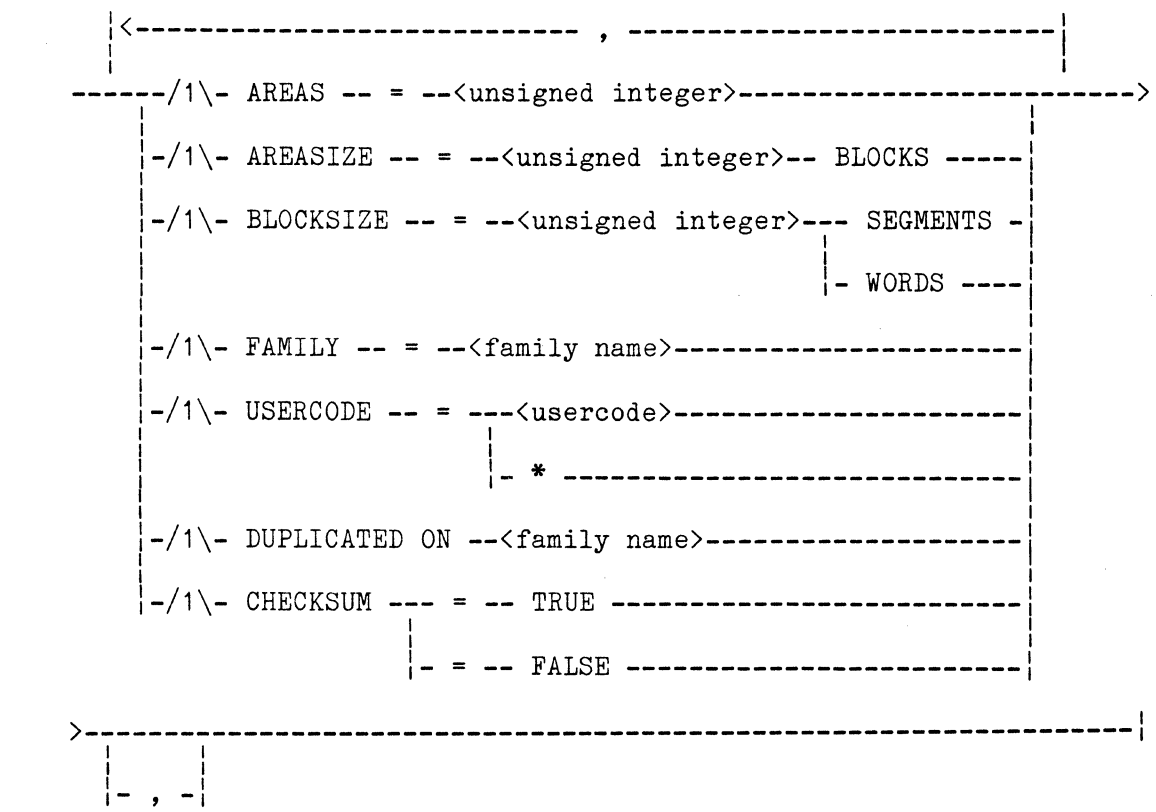
   This file contains control information including a Halt/Load flag to indicate that recovery is required. For each user known to a transaction journal, the control file contains the most recent response transaction record. This response record, in turn, contains a reference to either the user's last processed input transaction record or the user's last good restart transaction record. (The concept of a user known to a transaction journal is explained in greater detail in Section 5 of this manual.) The control file has a title of the form <base name>/<journal name>/CONTROL.

The TFL definition of a transaction journal gives the values of file attributes and other specifications for the control and data files that make up a particular transaction journal. For any transaction journal that is not explicitly defined, these specifications are given default values.

Syntax:


<transaction journal>
--<journal name>-- JOURNAL --
- TRANSACTION -
, --
>-/1\- CONTROL FILE -- (--<control file specs>--) --
-/1\- DATA FILE -- (--<data file specs>--) --
<journal name>
--<simple identifier>--
<control file specs>
, --
-/1\- AREAS -- = --<unsigned integer>--
-/1\- AREASIZE -- = --<unsigned integer>-- BLOCKS --
-/1\- BLOCKSIZE -- = --<unsigned integer>--- SEGMENTS -
- WORDS --
-/1\- FAMILY -- = --<family name>--
-/1\- USERCODE -- = ---<usercode>--
- * --
-/1\- CHECKSUM --- = -- TRUE --
- = -- FALSE --
>--
- , -

<data file specs>



Semantics:

<journal name> JOURNAL
<journal name> TRANSACTION JOURNAL

Each of these constructs identifies the definition of a transaction journal. <journal name> can be any valid <simple identifier>, except that if the journal is used for tanking it cannot be named TRHISTORY.

AREAS = <unsigned integer>
AREASIZE = <unsigned integer> BLOCKS
BLOCKSIZE = <unsigned integer> SEGMENTS
BLOCKSIZE = <unsigned integer> WORDS

These constructs assign the specified values to control file or data file attributes. These attributes are described in detail in the I/O Subsystem Reference Manual. The following list gives the system default attribute values.

Attribute	Default Value
AREAS	100
AREASIZE	30 Blocks
BLOCKSIZE	900 Words

FAMILY = <family name>

Causes the control or data file to be created on the specified family. <family name> is defined under "TFL Basic Constructs" in this section. If no FAMILY specification is given, DISK is the default.

USERCODE = <usercode>
USERCODE = *

These constructs cause the control or data file to be created under either the specified usercode or the system node (*). If no USERCODE specification is given, no usercode is assumed. In this situation, the system attempts to locate the appropriate file(s) under the usercode associated with the user-written program.

DUPLICATED ON <family name>

Causes a data file to be duplicated on the specified family. <family name> is defined under "TFL Basic Constructs" in this section. If no DUPLICATED specification is given, the file is not duplicated.

Only data files can be duplicated. This provision is made because data files contain original data that cannot otherwise be recovered if lost, whereas a journal control file can be recovered by the TRUTILITY program. (For more details about the TRUTILITY program, refer to Section 11 of this manual.)

If a USERCODE specification is given, the duplicated file as well as the primary file is created under that usercode.

CHECKSUM = TRUE
CHECKSUM = FALSE

Causes blocks of data in control files and data files to be checksummed. The CHECKSUM specification is used to detect I/O errors, and thus is useful in ensuring the integrity of journal files. A CHECKSUM is a value computed for each block by applying an equivalence operator to each word in the block. When the block is physically written, the CHECKSUM value is stored in a reserved word

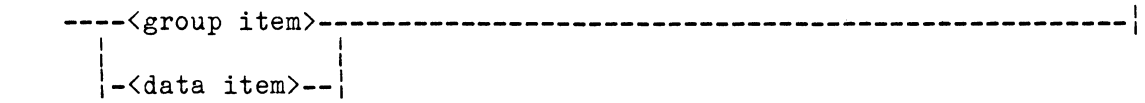
with the block. When the block is read, the CHECKSUM value is recomputed, and the result is compared to the stored value. A CHECKSUM error occurs if the two values are not equal. CHECKSUM is set TRUE by default.

Items

<item>s that can be defined for a <transaction record format> can be either <group item>s or <data item>s. <group item>s and <data item>s are explained in the following paragraphs.

Syntax:

<item>



Group Item

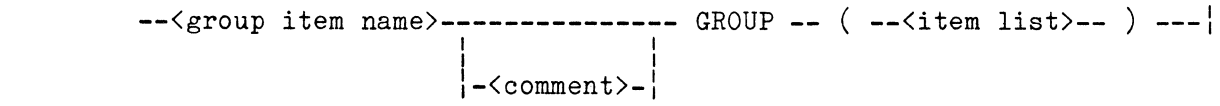
A <group item> is an <item> that contains other <item>s. When referenced as an <item>, a <group item> behaves like an <alpha item> of the same length. Unlike an <alpha item>, however, a <group item> cannot have an initial value of its own; its initial value is determined by the initial values of its elementary items.

If a <group item> is declared with an <occurs option>, then each <item> in the group inherits that <occurs option>. (<occurs option> is defined under "Data Item" in this section.)

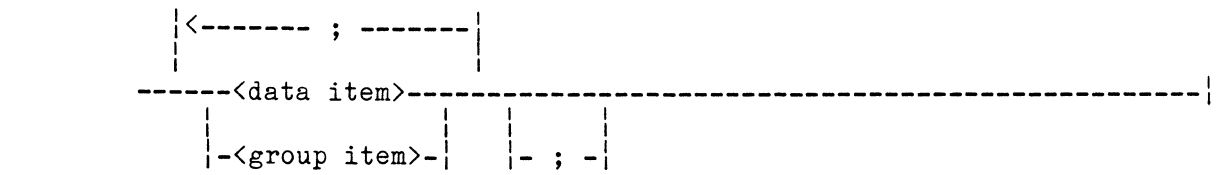
A <group item> always begins on a byte boundary. Whenever necessary, a binary zero digit is used as filler.

Syntax:

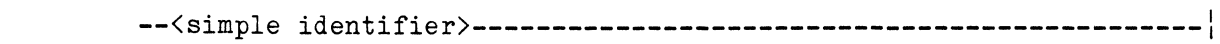
<group item>



<item list>



<group item name>



Semantics:

<group item name> GROUP

Identifies the declaration of a <group item>.

<simple identifier>
<comment>

Defined under "TFL Basic Constructs" in this section.

<item list>

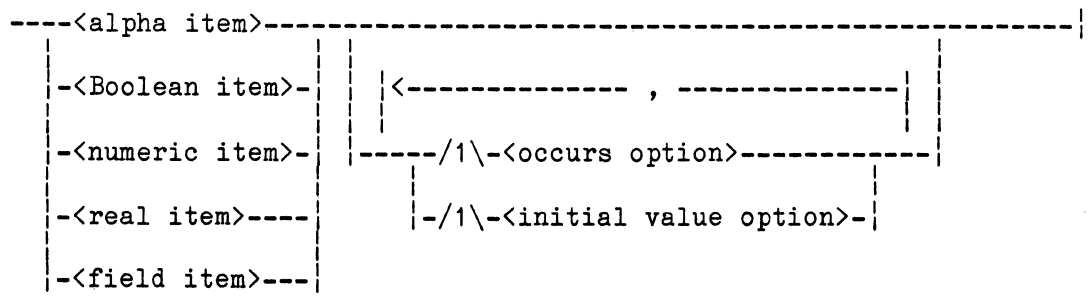
Defined under "Transaction Record Format" in this section.

Data Item

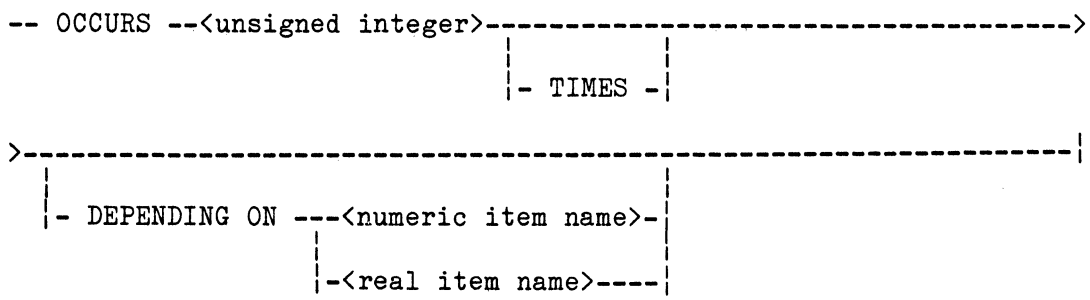
A <data item> is defined to be an <alpha item>, a <Boolean item>, a <numeric item>, a <real item>, or a <field item>. Items of each type are defined in the following paragraphs.

Syntax:

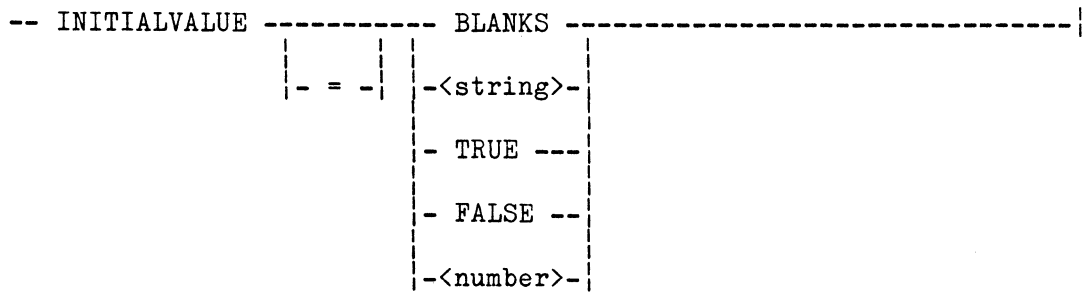
<data item>



<occurs option>



<initial value option>



Semantics:

OCCURS <unsigned integer>
OCCURS <unsigned integer> TIMES

These constructs cause the declared item to appear <unsigned integer> times within the transaction record. References to the item must be subscripted to indicate which occurrence of the item is intended. Valid subscripts range from 1 to the value of the <unsigned integer>. The value of the <unsigned integer> must not exceed 4095.

DEPENDING ON <real item name>
DEPENDING ON <numeric item name>

Causes data in the record to be compacted. Only the number of occurrences given by <real item name> or <numeric item name> in the record are retained when the record is written to a journal data file. The user program is responsible for setting the specified item to the proper value.

The item must be a non-subscripted <numeric item> or <real item> with no fractional part, and it must contain a positive integer. The declaration of the numeric item must precede the declaration of the <data item>, but need not be adjacent to it.

INITIALVALUE

Specifies the value that the <data item> is given when a record containing that item is created. The equal sign (=) is optional.

BLANKS or <string> can only be specified for <alpha item>s, TRUE or FALSE for <Boolean item>s, and <number> for <numeric item>s and <real item>s. <field item>s cannot be assigned an INITIALVALUE; they receive default INITIALVALUEs. When a <number> is specified, its value must be within the range of values permitted by the <precision> and <scale factor> of the item.

If INITIALVALUE is specified for an occurring item, then each occurrence of the item is initialized to the specified value. If no INITIALVALUE is assigned to an item, that item is assigned a default value depending on the item type (refer to <item defaults> under Defaults in this section). If no INITIALVALUE is assigned under <item defaults>, the system default value is assigned. System default INITIALVALUEs for items are as follows:

Type	Value
ALPHA	BLANKS
BOOLEAN	FALSE
NUMBER	0
REAL	0

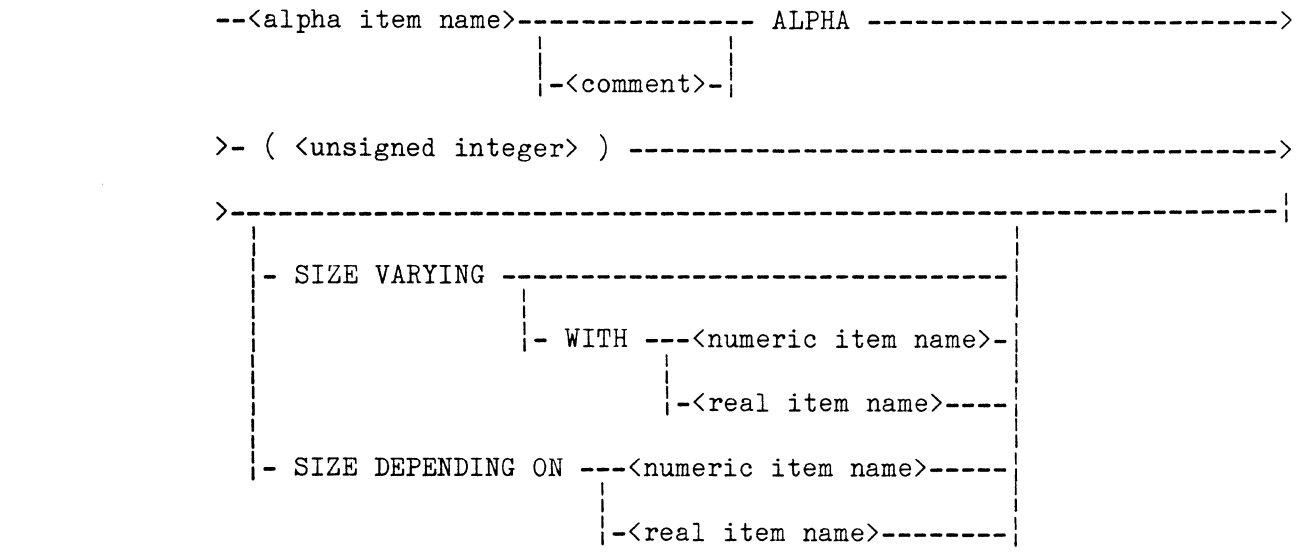
Alpha Item

An <alpha item> contains a string of up to 4095 EBCDIC characters aligned on a byte boundary. Whenever necessary, a binary zero digit is used as filler.

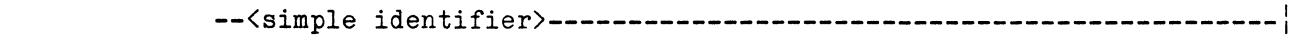
The default initial value for an alpha item is all blanks.

Syntax:

<alpha item>



<alpha item name>



Semantics:

<alpha item name> ALPHA (<unsigned integer>)

Declares an <alpha item> named <alpha item name> that can contain up to <unsigned integer> EBCDIC characters aligned on a byte boundary. <unsigned integer> must be in the range 1 to 4095.

SIZE VARYING
SIZE VARYING WITH <numeric item name>
SIZE VARYING WITH <real item name>

The SIZE option is used to conserve disk space when storing alpha information of varying lengths. If SIZE VARYING alone is specified, trailing blanks in the <alpha item> are eliminated before the transaction record is written to a transaction journal data file. If WITH <numeric item name> or WITH <real item name> is also specified, the system computes the length of the information in the <alpha item> and stores this value in the <numeric item> or <real item> following the WITH. The length is defined as the number of characters up to and including the last nonblank character.

<numeric item name> or <real item name> must refer to a nonsubscripted <numeric item> or <real item> with no functional parts. The <numeric item> must be associated with only one variable item. The declaration of the <numeric item> must precede the declaration of the <alpha item>, but need not be adjacent to it.

SIZE DEPENDING ON <numeric item name>
SIZE DEPENDING ON <real item name>

For "SIZE DEPENDING ON" items, the number of characters stored in the <alpha item> (when the transaction record is stored on disk) is the value of the given <numeric item> or <real item> or the declared size of the <alpha item>, whichever is less. The user is responsible for setting the <numeric item> or <real item> to the proper value. If the value of the <numeric item> or <real item> is 0, no characters are stored. <numeric item name> or <real item name> must refer to a nonsubscripted <numeric item> or <real item> with no fractional part, and must contain a positive integer.

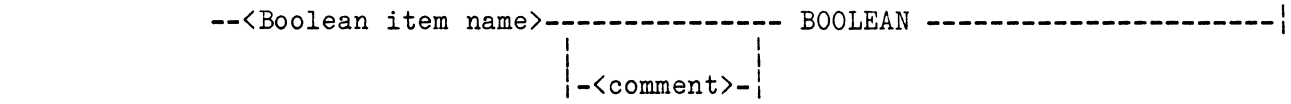
Several items using the SIZE DEPENDING ON option may depend on a single <numeric item>. The declaration of the <numeric item> must precede the declaration of the <alpha item>, but need not be adjacent to it.

Boolean Item

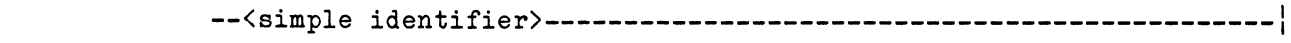
A <Boolean item> is a single 4-bit hex digit representing one of the truth values TRUE or FALSE. The default initial value is FALSE. The value TRUE is represented by 1 and the value FALSE by 0.

Syntax:

<Boolean item>



<Boolean item name>



Semantics:

<Boolean item name> BOOLEAN

Declares a <Boolean item> with the specified name.

<comment>
<simple identifier>

Defined under "TFL Basic Constructs" in this section.

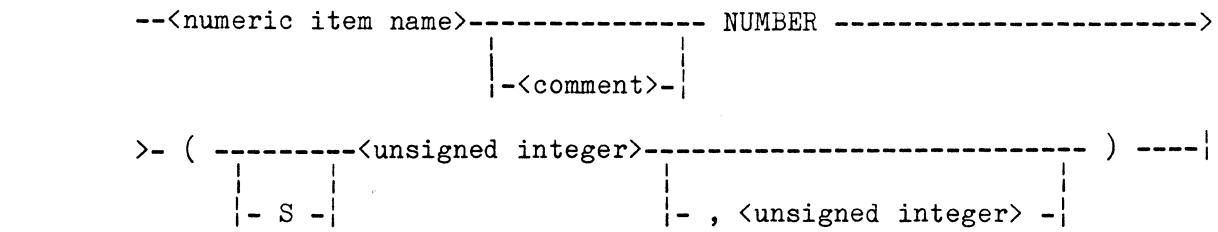
Number Item

A <numeric item> contains a string of 4-bit hex digits representing a signed or unsigned, fractional or integer value. Numbers are represented in 4-bit packed decimal form. The maximum size, including sign, is 23 digits.

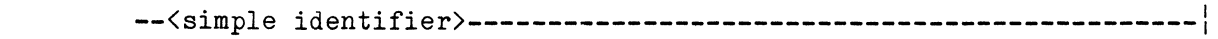
The system default initial value of a <numeric item> is 0.

Syntax:

<numeric item>



<numeric item name>



Semantics:

<numeric item name> NUMBER (<unsigned integer>)

Declares a numeric item with the specified name. The item contains a total of <unsigned integer> numeric digits. <unsigned integer> must be in the range 1 through 23.

<numeric item name> NUMBER (<unsigned integer>, <unsigned integer>)

Declares a numeric item with the specified name. The item contains the number of digits given by the first <unsigned integer>, and the precision given by the second <unsigned integer>. (That is, the number of digits to the right of the decimal point is given by the second <unsigned integer>.) The second <unsigned integer> must not exceed the first <unsigned integer>.

S

Causes a sign digit to be stored along with the value. If the S option appears, the size of the item is increased by one digit. If the S option does not appear, only the absolute value is stored. The maximum size for unsigned items is 23, and the maximum size for signed items is 22.

<comment>

Defined under "TFL Basic Constructs" in this section.

Real Item

<real item>s are used to store signed or unsigned fractional or integer values. Values are represented in one of two forms:

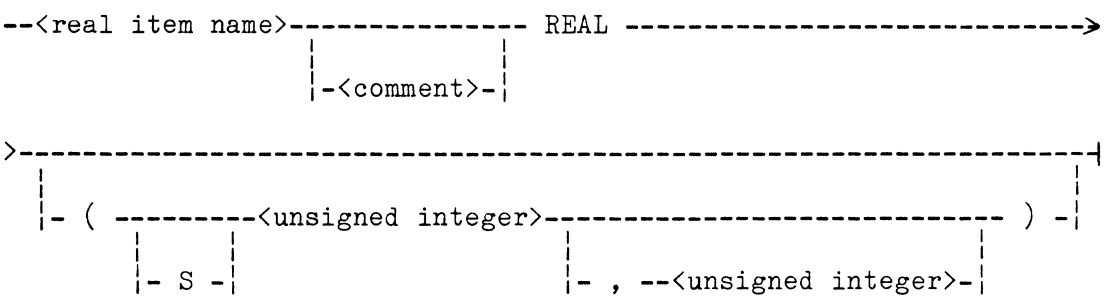
1. Binary, single-precision floating-point form.

2. Binary integer form.

The system default initial value for a <real item> is 0, and one word (twelve 4-bit digits) is always allocated for every <real item>. <real item>s are aligned on 8-bit byte boundaries. The maximum size of a <real item>, including sign, is 12 digits.

Syntax:

<real item>



<real item name>

Semantics:

<real item name> REAL

If no other options appear, declares a <real item> with the specified name in binary, single-precision floating-point form.

<real item name> REAL (<unsigned integer>)

Declares a <real item> with the specified name in binary integer form. The declared <real item> contains a total of <unsigned integer> numeric digits.

<real item name> REAL (<unsigned integer>, <unsigned integer>)

Declares a <real item> with the specified name in binary integer form. The declared <real item> contains the number of digits given by the first <unsigned integer>, to the precision given by the second <unsigned integer>. (That is, the number of digits to the right of the decimal point is given by the second <unsigned integer>.) The value of the second <unsigned integer> must not exceed the value of the first <unsigned integer>.

S

If the S appears before the first <unsigned integer>, the item includes a sign. If the S option does not appear, only the absolute value is stored. For unsigned items, the maximum value for the first <unsigned integer> is 12, and for signed items is 11.

<comment>

Defined under "TFL Basic Constructs" in this section.

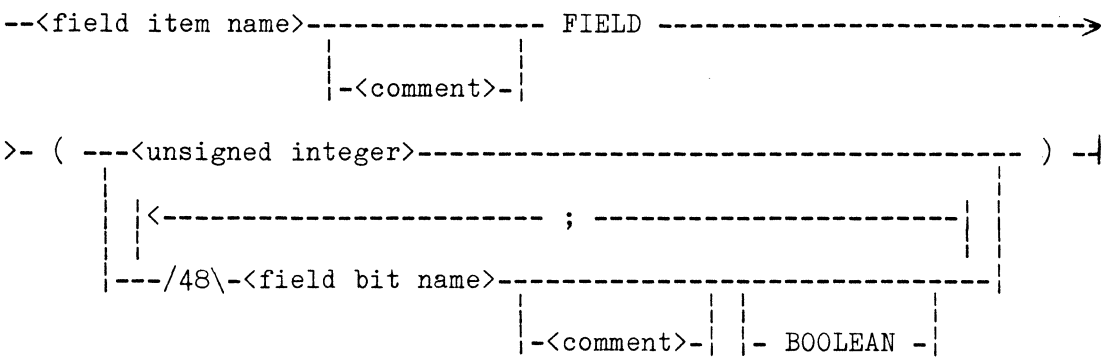
Field Item

A field item can be used to store unsigned integer and Boolean values. It represents unsigned integer values in binary form. A <field item> is a field of bits stored as a string of 4-bit hex digits. <field item>s always begin on a 4-bit digit boundary and occupy an integral number of 4-bit digits.

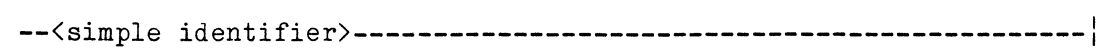
The system default initial value of a <field item> is all bits OFF or 0.

Syntax:

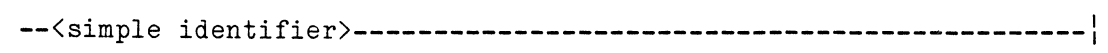
<field item>



<field item name>



<field bit name>



Semantics:

<field item name> FIELD (<unsigned integer>)

If no other options appear, declares a <field item> of the specified name, containing <unsigned integer> bits aligned on a 4-bit boundary. <unsigned integer> must be in the range 1 through 48. The maximum value that a <field item> can contain is controlled by <unsigned integer>. If <unsigned integer> is less than or equal to 39, then the field can contain an integer value between 0 and

(2**<unsigned integer>)-1. For values of <unsigned integer> greater than 39, the field can contain values between 0 and (2**39)-1.

<field item name> FIELD (<field bit name>; <field bit name> ...)

Declares a <field item> with the specified name containing the number of bits in the <field bit name> list. If a <field item> is declared in this way, each bit in the item can be accessed as a Boolean value by using the corresponding <field bit name>. This option can be used to minimize storage space when several Booleans are required.

APPEARANCE OF TFL ITEMS TO APPLICATION LANGUAGES

The following table shows how TFL items of each type must be declared in each application language for programs that access a transaction base. In each listing, <name> is the declared item name, and m and n are unsigned integers.

TFL Item	Application Language	
<name> ALPHA(n)	ALGOL:	STRING <name>
	COBOL74:	<name> PIC X(n) DISPLAY
	COBOL:	<name> PIC X(n) DISPLAY
<name> NUMBER(n)	ALGOL:	INTEGER <name>
	COBOL74:	<name> PIC 9(n) COMP
	COBOL:	<name> PIC 9(n) COMP-2
<name> NUMBER(Sn)	ALGOL:	INTEGER <name>
	COBOL74:	<name> PIC S9(n) COMP
	COBOL:	<name> PIC S9(n) COMP-2
<name> NUMBER(n,m)	ALGOL:	REAL <name>
	COBOL74:	<name> PIC 9(n-m)V9(m) COMP
	COBOL:	<name> PIC 9(n-m)V9(m) COMP-2
<name> NUMBER(Sn,m)	ALGOL:	REAL <name>
	COBOL74:	<name> PIC S9(n-m)V9(m) COMP
	COBOL:	<name> PIC S9(n-m)V9(m) COMP-2
<name> REAL	ALGOL:	REAL <name>
	COBOL74:	<name> BINARY
	COBOL:	<name> COMP-4
<name> REAL(n)	ALGOL:	INTEGER <name>
	COBOL74:	<name> 9(n) BINARY
	COBOL:	<name> PIC 9(n) COMP
<name> REAL(Sn)	ALGOL:	INTEGER <name>
	COBOL74:	<name> PIC S9(n) BINARY
	COBOL:	<name> PIC S9(n) COMP
<name> REAL(n,m)	ALGOL:	REAL <name>
	COBOL74:	<name> PIC 9(n-m)V9(m) BINARY
	COBOL:	<name> PIC 9(n-m)V9(m) COMP
<name> REAL(Sn,m)	ALGOL:	REAL <name>
	COBOL74:	<name> PIC S9(n-m)V9(m) BINARY
	COBOL:	<name> PIC S9(n-m)V9(m) COMP

Transaction Formatting Language (TFL)

<name> BOOLEAN	ALGOL:	BOOLEAN <name>
	COBOL74:	<name> BOOLEAN
	COBOL:	<name> BOOLEAN
<name> FIELD(n)	ALGOL:	REAL <name>
	COBOL74:	<name> FIELD SIZE IS n BITS
	COBOL:	<name> FIELD SIZE IS n BITS
<name> GROUP	ALGOL:	STRING <name>
	COBOL74:	<name>
	COBOL:	<name>

UPDATING TRANSACTION BASES

A transaction base can be modified by making changes to the TFL source file and generating a new TRDESCRIPTION file. Before running the TFL update, the UPDATE option must be inserted at the beginning of the revised TFL source file (as shown in TFL DESCRIPTION OF TRANSACTION BASE in this section), and the SYSTEM/TFL HOST file must be label-equated to the old TRDESCRIPTION file. (For more information regarding the files used by SYSTEM/TFL, refer to SYSTEM/TFL Input Files in this section.)

When the user makes changes to the items of a subformat or format, a new format is generated that invalidates the previous one. If the <format name> remains the same, then the new format has the same format number but can still be distinguished from the old format by the Transaction Library. If the <format name> is changed, or if a format is deleted and a new one added, then the old format number is invalidated and a new format number must be assigned.

The update level of a transaction base is incremented when a format is altered, added or deleted. If the TFL processor option ZIP is set, the Transaction Library is automatically recompiled when the update level is incremented. Although the user's Update Library is not automatically recompiled, it must match the update level of the Transaction Library. Therefore, the user is responsible for ensuring that the Update Library is recompiled also.

User-written programs that do not use the invalidated formats can continue to run without being recompiled. However, programs using an invalidated format receive an error if they pass an invalidated format to the Transaction Library.

Global parameters and options can also be changed. The DATABASE, STATISTICS, RESTARTDATASET, and HOSTSYSTEM <parameters> can also be changed through an update. (Refer to Parameters in this section.) The Transaction Library must be recompiled to incorporate the changes, but user-written programs need not be recompiled.

Specifications for the <default journal> and the <transaction journal>s can be updated in the same manner. The FAMILY, USERCODE and CHECKSUM <control file specs> file can be changed, but the AREAS, AREASIZE, and BLOCKSIZE attributes cannot. All <data file specs> can be changed, including the physical attributes. If the AREASIZE attribute or the CHECKSUM specification is changed by a TFL update, the Transaction Library creates a new file containing the new attribute values the first time the transaction base is opened, rather than continuing to add transactions to an existing data file.

A subbase can be changed by renaming it or by adding or deleting formats or subformats. Any subbase change deletes the previous subbase and adds a new one with a new subbase number. Any program creating a transaction record in an old subbase can pass the transaction through the Transaction Library as long as that format is still valid.

SYSTEM/TFL

SYSTEM/TFL is a program that functions as a processor. SYSTEM/TFL reads the symbolic description of a transaction base written in TFL and produces a description file named <base name>/TRDESCRIPTION. Depending on the TFL processor options specified by the user, SYSTEM/TFL can perform other functions as well, including the following:

1. SYSTEM/TFL can produce a printer listing of the TFL source file, including errors that have been detected.

2. If UPDATE is specified in the TFL source, SYSTEM/TFL can create an updated TRDESCRIPTION file.

3. If the TFL ZIP option is specified, SYSTEM/TFL can initiate compilation of all TPS tailored software.

SYSTEM/TFL Input Files

SYSTEM/TFL employs the following files, which are similar to compiler files:

CARD	CARD is the primary input file. It is required whenever SYSTEM/TFL is run. CARD is a card-image file that contains the entire TFL source description of a transaction base, unless the MERGE dollar-card option is set. In that case, CARD contains only patches to the TAPE file. The user should label-equate TAPE to the title of the primary input file.
TAPE	TAPE is the secondary input file. When the MERGE dollar-card option is set, TAPE contains the TFL source and CARD contains patches to the TFL source. The user should label-equate TAPE to the title of the secondary input file.
HOST	HOST is the previous TRDESCRIPTION file used as input when performing an update to the transaction base. The user should label-equate the HOST file to the title of the TRDESCRIPTION file.

CODE	CODE is the primary output file. It is the TRDESCRIPTION file that is produced if no errors are detected. Its title, <base name>/TRDESCRIPTION, is set by SYSTEM/TFL.
LINE	LINE is the printer output file. It contains various output, depending on the options set by the user.
NEWSOURCE	NEWSOURCE is the updated source file, produced when the dollar-card option NEW is set. The user should label-equate NEWSOURCE to the title of the updated source file.

Example:

The following example shows a sample SYSTEM/TFL run:

```
RUN SYSTEM/TFL;
FILE CARD = PATCH/TEST;
FILE TAPE = TFL/TEST;
FILE HOST = TEST/TRDESCRIPTION;
```

SYSTEM/TFL Dollar-Card Options

The dollar-card options listed in this section control the manner in which the TFL processor processes the source input. Dollar-card options can appear anywhere in the TFL source input. An explanation of how these options work is also provided in the following paragraphs.

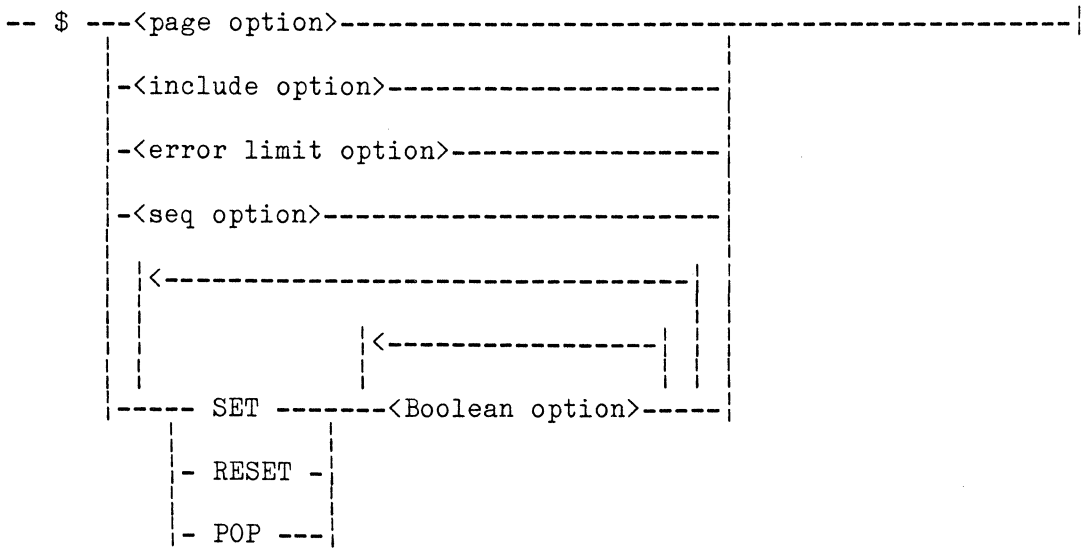
A 48-bit stack is allocated for each option. The top location in the stack contains the current value of the option, and the locations below it record the option's previous values. Every option has a default initial value. An option that is initially set is assigned a 48-bit stack with a 1 on top and zeros in the remaining positions; an option that is initially reset is assigned a 48-bit stack filled with zeros.

When an option is set, the option's stack is pushed down one bit and a 1 is placed on top of the stack. When an option is reset, the stack's contents are pushed down and a 0 is inserted on top of the stack. Because the stack only contains 48 bits, a history of at most 48 states can be recorded. If the stack is full when a SET or RESET is performed, the oldest state is simply discarded. POP causes the stack for the specified option to be popped up one bit; the current state of the option is discarded, and the option reverts to its previous state.

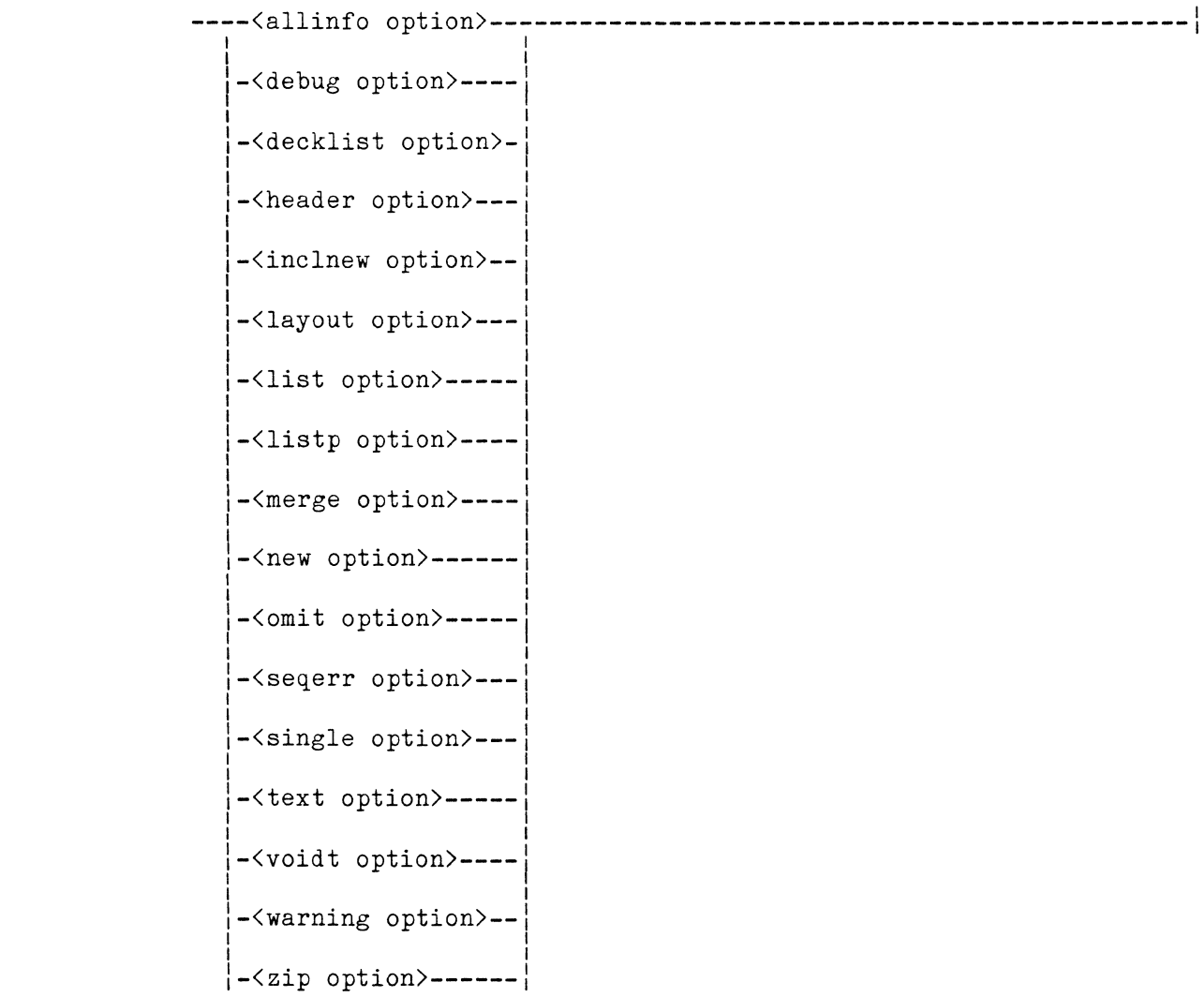
TFL dollar-card options assign values to Boolean options explicitly. The value of a Boolean option is changed when the option begins with the verb SET, RESET, or POP. Only the options listed following SET, RESET, or POP are changed. The values of all other options remain the same. Once an option is assigned a value, it retains that value until it is altered at a later time.

Syntax:

<dollar card option>



<Boolean option>



Example:

The following TFL dollar-card option sets the LIST and ALLINFO options. When the ALLINFO option is set, the TEXT, LAYOUT, and DECKLIST options are also set automatically.

```
$ SET ALLINFO LIST
```

The following list gives pertinent information about the TFL dollar-card options. For all <Boolean option>s, the initial default value is given in parentheses following the option name.

ALLINFO (RESET)

When the <allinfo option> is set, the <text option>, <layout option>, and <decklist option> are SET automatically.

Syntax:

<allinfo option>

```
-- ALLINFO ------------------------------------------------------¦
```

DEBUG (RESET)

The <debug option> is an internal debugging aid. When the <debug option> is set, TFL produces additional line printer output that is used during system development.

Syntax:

<debug option>

```
-- DEBUG --------------------------------------------------------¦
```

DECKLIST (RESET)

When the <decklist option> is set, the job file that is ZIPped to compile TPS software is listed.

Syntax:

<decklist option>

```
-- DECKLIST -----------------------------------------------------¦
```

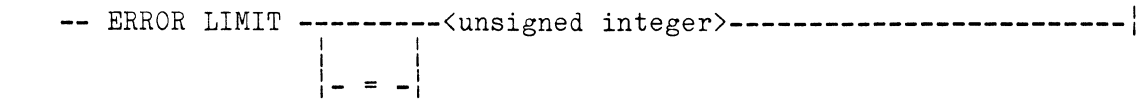
ERROR LIMIT

The <error limit option> controls the number of syntax errors that SYSTEM/TFL allows before it stops processing. If the number of errors equals or exceeds the maximum allowable value, processing is stopped. A value of 0 for this option allows any number of errors.

The default value of the <error limit option> is 10 if SYSTEM/TFL is run through CANDE and 100 otherwise.

Syntax:

<error limit option>



Semantics:

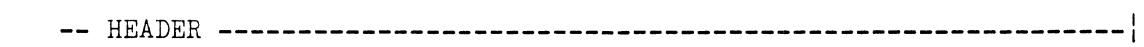
<unsigned integer>

Gives the maximum number of syntax errors allowed for a SYSTEM/TFL run. The equal sign (=) is optional.

HEADER (RESET)

When the <header option> is set and the <list option> is reset, only the standard header information and a compilation summary are listed. Source input is not listed.

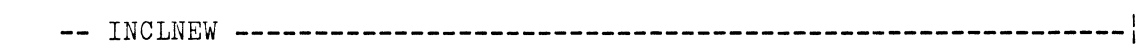
Syntax:



INCLNEW (RESET)

When the <inclnew option> is set, the source from an INCLUDEd file is written to the NEWSOURCE file in place of the <include option> statement. If the <new option> is reset, the <inclnew option> is ignored.

Syntax:



INCLUDE

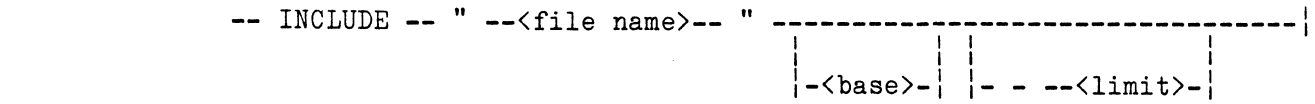
The <include option> includes source language input from files other than CARD and TAPE files. Each <include option> specifies the title of the input file in quotes and indicates which portion of the file is to be included. The included card images are processed in place of the <include option>.

<include option>s can appear anywhere in the input. However, <include option>s must not be nested; that is, included input must not contain <include option>s.

Included input must have the same record size and format as the CARD file.

Syntax:

<include option>



Semantics:

INCLUDE "<file name>"

Specifies the name of the file to be included. If no other options appear, the entire file is included.

<base>

Specifies the sequence number of the first record to be included. If no <base> appears, the first record of the file is the beginning of the included range.

<limit>

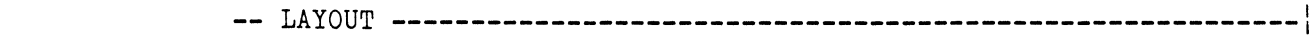
Specifies the sequence number of the last record to be included. If no <limit> is indicated, the last record of the file is the end of the included range.

LAYOUT (RESET)

When the <layout option> is set, all items along with their offsets and sizes are listed on the line printer.

Syntax:

<layout option>



LIST (RESET if run through CANDE; otherwise, SET)

When the <list option> is set, source input is listed on the line printer. When the <list option> is reset, only syntax errors are listed.

Syntax:

<list option>

-- LIST ----------¦

LISTP (RESET)

When the <listp option> is set, CARD input is listed but TAPE input is not. The <listp option> is ignored when the <list option> is set.

Syntax:

<listp option>

-- LISTP ----------¦

MERGE (RESET)

When <merge option> is set, input from the CARD file is merged with input from the TAPE file. If a CARD record has the same sequence number as a TAPE record, the CARD record replaces the TAPE record. When the <merge option> is reset, only the CARD file is used as input.

Syntax:

<merge option>

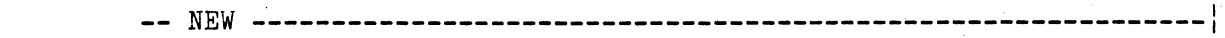
-- MERGE ----------¦

NEW (RESET)

When the <new option> is set, the updated symbolic file NEWSOURCE is created. This file can be used as the TAPE input file during subsequent TFL runs. The NEWSOURCE file is created whether or not syntax errors are present in the input.

Syntax:

<new option>

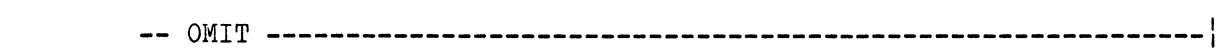


OMIT (RESET)

When the <omit option> is set, input from both CARD and TAPE is ignored. The omitted records are written to the NEWSOURCE file if the <new option> is set or to the LINE file if the <list option> is set. However, the omitted statements are not processed. Omitted records are denoted by the word OMIT on the listing.

Syntax:

<omit option>

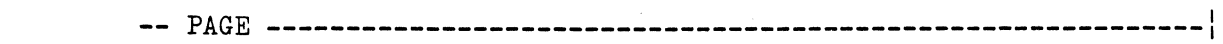


PAGE

If the <list option> or <listp option> is set, the <page option> causes a skip to the top of a new page.

Syntax:

<page option>

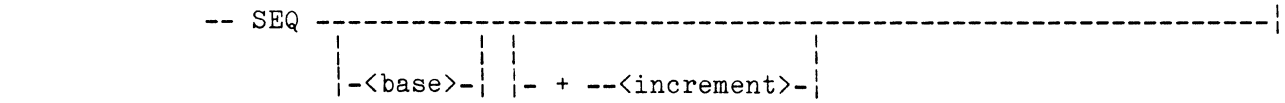


SEQ

When the <seq option> is set, the output file and the NEWSOURCE file are resequenced. When the <seq option> is reset, the original input sequence numbers are retained.

Syntax:

<seq option>



<base>

<increment>

Semantics:

SEQ

If no <base> or <increment> is given, SEQ causes sequencing to begin at 1000 and continue in increments of 1000.

<base>

Specifies the sequence number of the first record. If no <base> appears, sequencing begins at 1000.

+ <increment>

Specifies the increment to be used for sequencing. If no <increment> appears, the sequence increment is 1000.

SEQERR (RESET)

When the <seqerr option> is set, sequence errors in the input TAPE file are flagged as syntax errors. If sequence errors are detected, the TRDESCRIPTION file is not locked, and sequence error messages appear on the listing.

Syntax:

<seqerr option>

```
-- SEQERR ------------------------------------------------------¦
```

SINGLE (SET)

When the <single option> is set, printer output is single-spaced. Otherwise, output is double-spaced.

Syntax:

```
-- SINGLE ------------------------------------------------------¦
```

TEXT (RESET)

When the <text option> is set, all TFL-generated source statements that are included in the TPS software are listed.

Syntax:

<text option>

```
-- TEXT --------------------------------------------------------¦
```

VOIDT (RESET)

If the <voidt option> is set, all secondary input from the TAPE file is ignored until the <voidt option> is reset. Therefore, while the <voidt option> is TRUE, only primary input is processed. The ignored input is neither listed nor included in the updated source file, regardless of the values of the <list option> and the <new option>. Once the <voidt option> is set, it can be reset or POPed only by an option in the CARD file.

Syntax:

<voidt option>

```
-- VOIDT -------------------------------------------------------¦
```

WARNING (RESET)

When the <warning option> is set, a warning message is issued for each identifier in the TFL source that is a COBOL reserved word.

Syntax:

<warning option>

ZIP (SET)

When the <zip option> is set and no syntax errors are present, TFL ZIPs a WFL file titled <base name>/TRDESCRIPTION to compile all tailored TPS software.

Syntax:

<zip option>

4 USER LANGUAGE INTERFACE TO TPS

The user language interface to TPS consists of extensions implemented in the DMSII user languages ALGOL, COBOL74 and COBOL that provide access to a transaction base. This section gives syntax and semantics for the extensions in each interface language, which permit the user to invoke a transaction base, declare and manipulate transaction records, and pass transaction records to procedures in the Transaction Library (described in the Section 5 of this manual). This section also provides skeleton user-written programs in ALGOL and COBOL that demonstrate the declaration of Transaction Library entry points in these languages.

The user written programs that invoke the Transaction Library should not be libraries themselves.

In the following constructs for the different languages, all references to an identifier are language-sensitive. That is, the restrictions for an identifier vary depending on the particular language being discussed. For details regarding the syntax of an identifier in any of the languages, the reader should refer to the appropriate language manual.

COBOL/COBOL74 USER LANGUAGE INTERFACE TO TPS

The following TPS extensions to the COBOL and COBOL74 languages are discussed under separate headings in this section.

1. Invocation of the transaction base.
2. Declaration of transaction record variables.
3. Creation of transaction records.
4. Accessing transaction record items.
5. Transaction record control items.
6. Transaction record compile-time functions.
7. Transaction record variables as parameters.
8. COBOL74-specific constructs for parameter passing.
9. Transaction record variable assignment.
10. Transaction use procedures.
11. Sample user-written program.

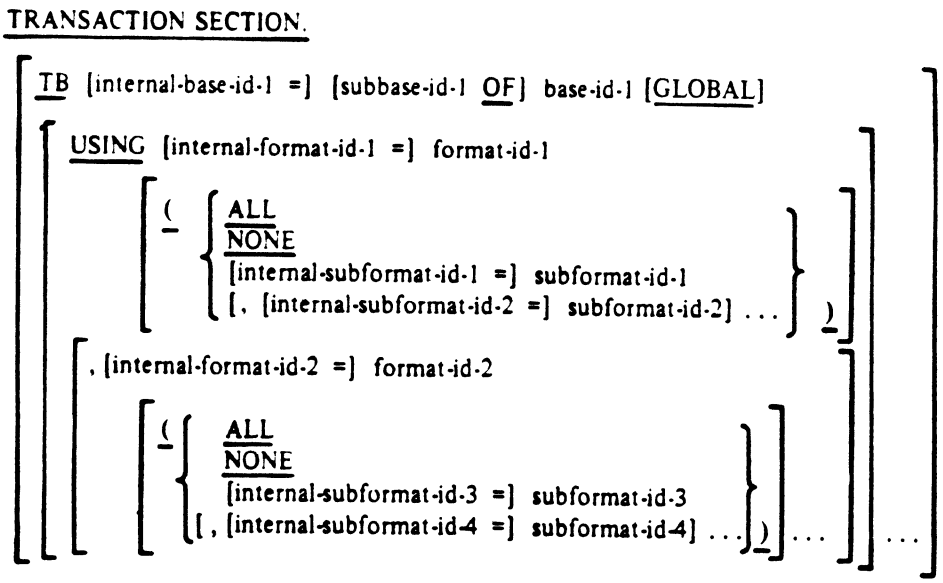
Invocation of Transaction Base

A user program must invoke a transaction base before making any references to formats or items defined within that transaction base. Invocation consists of specifying the transaction base to be invoked, and optionally, specifying a list of transaction record formats and subformats to be invoked. The program may also specify alternate internal names for the transaction base and for any of the formats or subformats invoked. If alternate internal names are specified for the base name, subbase name, format name, or subformat name, then the user-written program must reference these internal identifiers rather than the TFL source identifiers.

A user-written program can also invoke a subbase that has been defined for the transaction base. If a subbase is invoked, then only the transaction record formats and subformats defined within that subbase are accessible to the user-written program. As in transaction base invocation, the user program can specify a list of transaction record formats and subformats, using internal names if necessary, that can be invoked from that subbase.

A new section in the DATA DIVISION named the TRANSACTION SECTION contains the invocations of all transaction bases that can be referenced in a COBOL program. The TRANSACTION SECTION must appear between the DATABASE SECTION and the WORKING-STORAGE SECTION.

Syntax:



Semantics:

TRANSACTION SECTION

Invokes all transaction bases that can be referenced in a user-written program.

TB

Specifies the name of the transaction base or transaction subbase to be invoked in the user-written program. All transaction bases that can be referenced in the user-written program must have TB declarations.

internal-base-id

If internal-base-id is specified, the user-written program must reference this internal-base-id rather than the base-id.

subbase-id OF

If subbase-id is specified, only the transaction record formats and subformats defined within that subbase are accessible.

base-id

Specifies the TFL base name of the transaction base to be invoked.

USING

Specifies one or more transaction record formats and subformats to be invoked. If this option is specified, only the indicated transaction record formats and subformats are invoked. If this option is not specified, all transaction record formats and subformats are invoked for that transaction base or subbase.

internal-format-id

If optional internal-format-ids are specified, the user-written program must reference these internal-format-ids rather than the format-ids.

format-id

Specifies the name of the transaction record format being invoked.

ALL

If the reserved word ALL is used in the subformat-list, then all of the subformats of that format will be invoked.

NONE

If the reserved word NONE is used in the subformat-list, then no subformats of that format will be invoked.

internal-subformat-id

If optional internal-subformat-ids are specified, the user-written program must reference these internal-subformat-ids rather than the subformat-ids.

subformat-id

If a list of subformat-ids is specified, only those subformats are invoked. If this list is not specified, then all subformats of the transaction record format are being invoked.

GLOBAL

Declaring a transaction base to be global allows a separately compiled bound procedure to reference the same transaction base declared in the host program. Note the following:

1. The same transaction base formats and subformats must be declared in the same order in the two programs being bound. The logical identifiers for transaction bases, formats, and subformats must be the same in the two programs being bound.

2. Any entry point of a transaction library that is used in the bound procedure must be declared in the host program. This means that a COBOL74 host program must have CALL statements to all the transaction library entry points that will be used in the bound procedure.

3. If the host program is ALGOL and the bound procedure is COBOL74, the internal name (INTNAME attribute) must be HOSTLIB.

Declaration of Transaction Record Variables

A transaction record is an array row that can contain the transaction data of one of the transaction record formats declared in the TFL source. A transaction record variable names one of these array rows. By allowing a transaction record variable to contain the transaction data of one of the transaction record formats, the record becomes, in effect, a structured variable.

A transaction record variable can only be associated with one transaction base or transaction subbase and can contain only one of the formats and subformats that have been invoked for that particular transaction base or transaction subbase. The size of the array row is large enough to accommodate the largest of all the formats invoked for it.

Each transaction record for a given transaction base is declared using COBOL's 01 level indicator immediately after the appropriate TB entry in the TRANSACTION SECTION. Declaring the transaction records immediately following a TB entry associates those records with that particular TB entry.

Syntax:

```
01      transaction-record-id-1   [ OCCURS integer-1  [ TIMES ] ]  [GLOBAL]
[01     transaction-record-id-2   [ OCCURS integer-2  [ TIMES ] ]  [GLOBAL] ]
```

Semantics:

transaction-record-id

A simple COBOL/COBOL74 identifier that is used to name a transaction record variable.

transaction-record-id OCCURS integer
transaction-record-id OCCURS integer TIMES

Declares the indicated number of occurrences of a transaction record. The key word "TIMES" is optional. The transaction-record-id can be considered an array of transaction records. A specific transaction record within transaction-record-id can be selected by subscripting the transaction- record-id. Refer to Accessing Transaction Record Items in this section for more information about subscripting.

GLOBAL

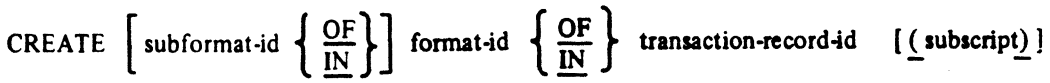
A transaction record or transaction record array that is declared in the host program can be declared global in the bound procedure using the GLOBAL clause. The GLOBAL clause cannot be specified in the host program; it is allowed only in COBOL74 procedures compiled at LEVEL3 or higher. If a transaction base is declared global, the associated transaction records are not made global unless they too have a global clause.

Creation of Transaction Records

The contents of a transaction record variable are undefined until the variable is initialized to a particular format by a CREATE statement. CREATE assigns the initial values of all items in the transaction record format (and subformat, if given) to the record variable and initializes the transaction record control items. When a format is created, only those items in the common part are assigned initial values. When a subformat is created, the common part items as well as the subformat part items are assigned initial values. The record variable continues to contain the given format until it is reinitialized by a subsequent CREATE. It is never cleared by the system.

Once a transaction record variable has been created in a particular format, and optionally, subformat, the items defined within the format and subformat can be accessed and manipulated. If a transaction record is created in a particular format, the record only contains the data items associated with that format. If a transaction record is created in a particular format and subformat, then the record contains the data items associated with the format and the data items associated with the subformat.

Syntax:



Semantics:

CREATE format-id OF transaction-record-id

The format-id is the name of the format whose data items' initial values are assigned to the transaction record variable. The transaction-record-id identifies the transaction record variable that is initialized to the particular format, and possibly the subformat if specified.

subformat-id

Subformat-id is the name of the subformat whose data items' initial values are assigned to the transaction record variable.

subscript

A COBOL/COBOL74 arithmetic expression that identifies a particular element within a table (declared with an OCCURS clause). If transaction-record-id is declared with the OCCURS option, then the subscript must appear.

Accessing Transaction Record Items

A transaction record can contain a transaction of any format or subformat that is declared in the TFL source. Data items in the declared format and subformat of that transaction can be referenced.

Transaction section data items are considered normal data items and can be referenced at any place in the PROCEDURE DIVISION that is acceptable for a normal data item.

Several special rules apply to the use of the CORRESPONDING phrase of a MOVE, ADD, or SUBTRACT statement in referencing TRANSACTION SECTION items. A transaction record name alone is not considered legitimate as a group name. Instead, a transaction record format name, a transaction subformat name, or a subordinate group data item must be referenced. When a transaction record format is referenced, only the data items in the common portion of the format are considered eligible for CORRESPONDING.

Syntax:

```
transaction-record-item-id { IN } [ subformat-id { IN } ] [ format-id { IN } ] transaction-record-id
                           { OF }                { OF }                { OF }

  [(subscript-1 [,subscript-2] ...)]
```

Semantics:

transaction-record-item-id

Identifies a particular transaction format data item that is to be referenced.

transaction-record-id

Identifies the transaction record variable that contains the data item to be referenced.

subformat-id

Identifies a particular transaction subformat. For qualification purposes, subformat-id may need to be specified in order to access a data item within a particular transaction record.

format-id

Identifies a particular transaction format. For qualification purposes, format-id may need to be specified in order to access a data item within a particular transaction record.

subscript

A COBOL/COBOL74 arithmetic expression that identifies a particular element within a table (declared with an OCCURS clause). If the transaction-record-id was declared with an OCCURS clause, a subscript is required, and it must be specified first if there are several subscripts.

The following must be subscripted:

1. Data items of transaction record formats or subformats that are occurring items;

2. Items embedded within one or more occurring groups;

3. Or occurring items embedded within occurring groups.

The subscripts within a list of transaction record items are listed from left to right, from the outermost occurring GROUP to the innermost occurring group or occurring items. They follow the subscript for the transaction record, is such a subscript is present.

A valid transaction-record-item-subscript ranges from 1 to the value of the unsigned-integer, which must not exceed 4095.

Example:

Subscripting is illustrated by the following example, using a TFL fragment of a transaction record format. The fourth item of E, in the second GROUP of C, in the seventh GROUP of A would be referenced with the subscript triple (7,2,4). Referencing data item D in the eighth GROUP of C, and the first GROUP of A would require the subscript pair (1,8).

```
 A    GROUP
(
   B  REAL;
   C  GROUP
   (
     D REAL;
     E NUMBER(6) OCCURS 5 TIMES;
   ) OCCURS 10 TIMES;
   ) OCCURS 20 TIMES;
```

When included in a COBOL program the above description looks like the following:

```
04 A OCCURS 20.
   05 B COMP-4.
   05 C OCCURS 10.
      06 D COMP-4.
      06 E PIC 9(6) COMP-2 OCCURS 5.
```

As another example, the third occurrence of the transaction record TR (as declared with an OCCURS clause), in the fourth item of E, in the second GROUP of C, in the seventh GROUP of A, can be referenced with the following syntax:

```
E OF TR (3,7,2,4)
```

Requirements For Data Item Qualification

The amount of qualification required to access a data item of a particular transaction record format or subformat varies. In every case, however, the transaction record variable containing the desired data item must be referenced. The following cases show the varying requirements for qualification:

Case 1. Data Item Qualification

If the name of the desired data item is unique with respect to data items of other invoked formats, then only the data item name need be specified.

Example:

```
DATAITEMNAME OF TRANREC
```

Case 2. Format Name and Data Item Name Qualification

If the name of the desired data item is not unique with respect to data items of other invoked formats, but is unique to the format that contains it, then both the format and the data item name are needed for qualification.

Example:

```
DATAITEMNAME OF FORMATNAME IN TRANREC
```

Case 3. Subformat Name and Data Item Name Qualification

Both the the subformat name and the data item name are needed for qualification if the following conditions prevail:

1. The name of the desired data item is not unique with respect to the common portion of another invoked format.
2. The data item is contained within a subformat.

3. Another data item within a different subformat of the same format has the same name as the desired data item.

Also, if the desired data item is contained within a subformat whose name is unique to all invoked formats and subformats, and the desired data item is not unique with respect to a subformat of another format, then both the subformat name and the data item name are needed for qualification.

Example:

DATAITEMNAME OF SUBFORMATNAME IN TRANREC

Case 4. Format Name, Subformat Name, & Data Item Name Qualification

The format name, subformat name, and data item name are all needed for qualification when the following conditions prevail:

1. The name of the desired item is not unique with respect to a subformat of another invoked format.

2. The name of the desired data item is not unique with respect to the format that contains it.

3. The name of the subformat that contains the desired data item is not unique with respect to all invoked formats and subformats.

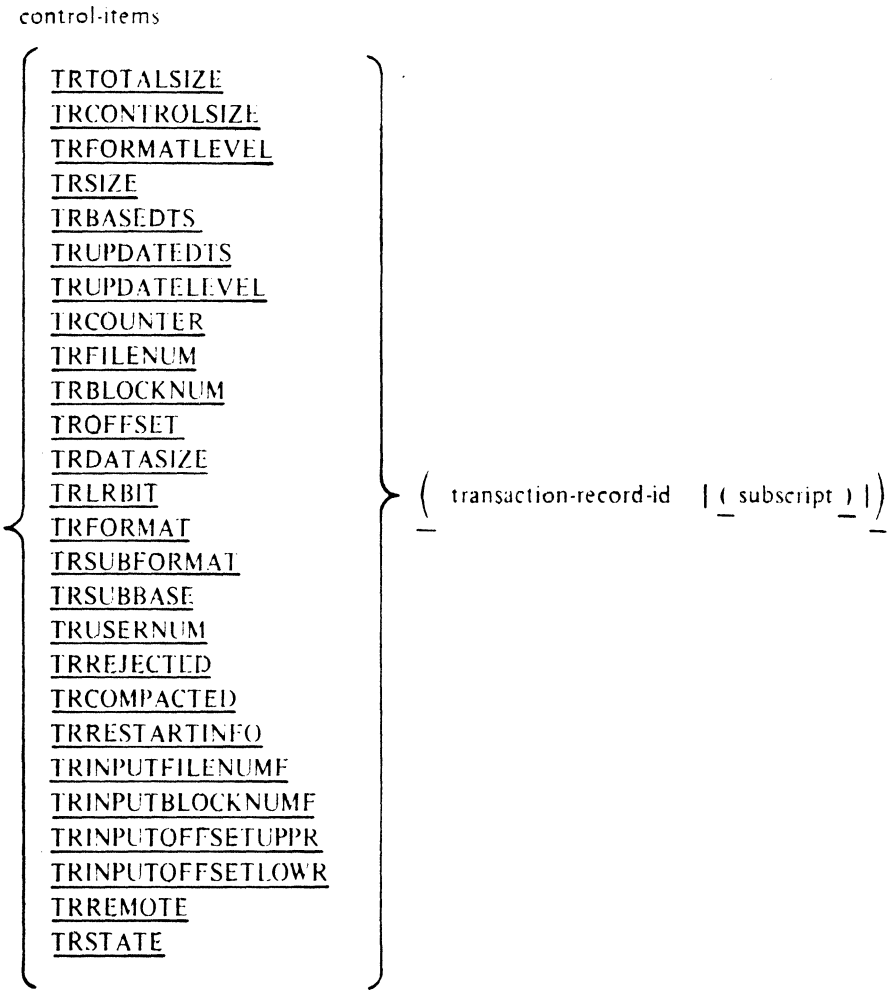
Example:

DATAITEMNAME OF SUBFORMATNAME OF FORMATNAME IN TRANREC

Transaction Record Control Items

Control items are system-defined items contained in every transaction record. These items are maintained by the Transaction Processing System and are read-only in all BDMSCOBOL and BDMSCOBOL74 programs. The initial values of these control items are assigned when a transaction record is created. These items are defined only after a transaction record has been created with a particular format and possibly subformat.

Syntax:


control-items
TRTOTALSIZE
TRCONTROLSIZE
TRFORMATLEVEL
TRSIZE
TRBASEDTS
TRUPDATEDTS
TRUPDATELEVEL
TRCOUNTER
TRFILENUM
TRBLOCKNUM
TROFFSET
TRDATASIZE
TRLRBIT
TRFORMAT
TRSUBFORMAT
TRSUBBASE
TRUSERNUM
TRREJECTED
TRCOMPACTED
TRRESTARTINFO
TRINPUTFILENUMF
TRINPUTBLOCKNUMF
TRINPUTOFFSETUPPR
TRINPUTOFFSETLOWR
TRREMOTE
TRSTATE
(transaction-record-id | (subscript) |)

Semantics:

TRTOTALSIZE

A numeric item that specifies the total size, in bytes, of the transaction record, including control and data parts. The size of the control part is a constant for all records.

TRCONTROLSIZE

A numeric item that specifies the size, in bytes, of the control portion of the transaction record. This size is fixed for all transaction records.

TRFORMATLEVEL

A numeric item that specifies the format level of the transaction record layout, defined as 1.

TRBASEDTS

A numeric item that contains the creation date-time stamp (with the TIME(6) value as defined in the ALGOL Reference Manual) associated with the TRDESCRIPTION file. TRBASEDTS is used to coordinate the TRDESCRIPTION file, user programs, and the Transaction Library.

TRSIZE (for COBOL only)

A numeric item that contains the actual size, in bytes, of a transaction record array row.

TRUPDATEDTS

A numeric item that specifies the update date-time stamp that defines the update level relative to the version of the TRDESCRIPTION used to compile a program. TRUPDATEDTS is also used to coordinate the various code files.

TRUPDATELEVEL

A numeric item that specifies the TFL update level, which is used to help coordinate TPS code files and user program code files. TRUPDATELEVEL is used when comparing transaction base update time stamps.

TRCOUNTER

A numeric item that contains a unique numeric value that synchronizes input and response transaction records between a remote system and the host system.

TRFILENUM
TRBLOCKNUM
TROFFSET

TRFILENUM, TRBLOCKNUM, and TROFFSET are all numeric items. Each refers to the record address of a transaction record in a journal data file. Each is assigned by the Transaction Library when the transaction is first written to a transaction journal data file.

TRFILENUM gives the number of the journal data file. TRBLOCKNUM gives the block number (in other words, the beginning segment address) within that file, and TROFFSET gives the word offset within that block.

TRDATASIZE

A numeric item that specifies the length, in bytes, of data items in a transaction record.

TRLRBIT

A bit used to distinguish a left control word from a right control word. TRLRBIT is defined as TRUE for a left control word; it is a numeric item.

TRFORMAT

A numeric item that contains a unique numeric value that identifies the transaction record format contained within a transaction record.

TRSUBFORMAT

A numeric item that contains a unique numeric value that identifies the transaction subformat contained within a transaction record. The value of TRSUBFORMAT is unique for a given TRFORMAT value.

TRSUBBASE

A numeric item that contains a unique numeric value that identifies the transaction subbase that has been invoked.

TRUSERNUM

A numeric item that identifies the source or owner of a transaction record, primarily to aid in recovery and restart after an interruption in processing. The last good input record and its response record are saved for each value of TRUSERNUM.

TRREJECTED

A Boolean item that is used by the Transaction Library to mark a transaction record that has already been written to the history file, and is subsequently rejected by the system when faults occur during processing.

TRCOMPACTED

A Boolean item that is used by the Transaction Library to mark a transaction record that is in its compacted form. A transaction record that cannot be compacted has a value of 0. The TRCOMPACTED field may hold the following values:

Value	Meaning
0	Transaction record cannot be compacted.
2	Transaction record can be compacted, but is not in compacted form.
3	Transaction record can be compacted and is in compacted form.

TRRESTARTINFO

A Boolean item that is used by the Transaction Library to designate how the transaction record that is contained within a transaction journal is written to the journal. The transaction record is written in one of the following ways, depending on the TRRESTARTINFO value:

Value	Meaning
0	No restart record follows this input transaction record.
2	A restart record follows this input transaction record, where the transaction record is the input record of the input-restart record pair.

1 A restart record follows this input transaction record, where the transaction record is the restart record of input-restart record pair.

TRINPUTFILENUMF
TRINPUTBLOCKNUMF
TRINPUTOFFSETUPPR
TRINPUTOFFSETLOWR

These control items are reserved for system use and refer to the addresses of input records in the transaction journal data file. Each of these control items represents a numeric value.

TRREMOTE

TRREMOTE is a Boolean item. It is a field that is used to distinguish between a record submitted from a remote system and a record submitted from the host system. If a transaction record is submitted from a remote system, TRREMOTE is defined to be TRUE.

TRSTATE

A numeric item that specifies the state of a transaction record during its processing period. TRSTATE is used to check whether a transaction record has passed through the appropriate states during its processing period.

transaction-record-id

Identifies a transaction record variable. This variable must be fully subscripted if a transaction record array element is used.

subscript

A COBOL/COBOL74 arithmetic expression that identifies a particular transaction record variable within an array of transaction record variables. If transaction-record-id is declared with the OCCURS option, then the subscript must appear.

transaction-record-id

Identifies a transaction record variable. This variable must be fully subscripted if a transaction record array element is used.

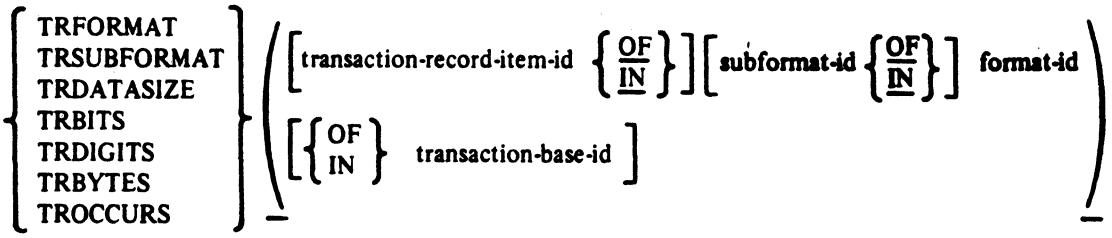
subscript

A COBOL/COBOL74 arithmetic expression that identifies a particular element within a table (declared with an OCCURS clause). If transaction-record-id is declared with the OCCURS option, then the subscript must appear.

Transaction Compile-Time Functions

Transaction compile-time functions provide access to certain properties of transaction record formats that are constant at compile time. These compile-time constructs are particularly useful when coding an Update Library. For examples of the use of compile-time functions, the reader should refer to Section 6 of this manual.

Syntax:



Semantics:

transaction-record-item-id

The name of a data item contained within an invoked transaction format or subformat.

subformat-id

The name of a transaction subformat that has been invoked within the program.

format-id

The name of a transaction format that has been invoked within the program.

transaction-base-id

The name of a transaction base that has been invoked within the program.

The following list specifies which transaction compile-time functions apply to the possible arguments, and what each function represents. Not all functions apply to all arguments. Transaction-base-id needs to be referenced only when transaction base qualification is required.

TRFORMAT format-id

Returns the numeric value assigned to the format-id.

TRSUBFORMAT subformat-id OF format-id
TRSUBFORMAT subformat-id IN format-id

Returns the numeric value assigned to the subformat-id.

TRDATASIZE format-id

Returns the size, in bytes, of the common portion of the transaction record.

TRDATASIZE subformat-id OF format-id
TRDATASIZE subformat-id IN format-id

Returns the size, in bytes, of the subformat record (including the common part).

```
TRBITS transaction-record-item-id OF format-id
TRBITS transaction-record-item-id IN format-id
TRBITS transaction-record-item-id OF subformat-id OF format-id
TRBITS transaction-record-item-id IN subformat-id IN format-id
```

Returns the size, in bits, of the referenced data item.

```
TRDIGITS transaction-record-item-id OF format-id
TRDIGITS transaction-record item-id IN format-id
TRDIGITS transaction-record-item-id OF subformat-id OF format-id
TRDIGITS transaction-record-item-id IN subformat-id IN format-id
```

Returns the size, in digits, of the referenced data item.

```
TRBYTES transaction-record-item-id OF format-id
TRBYTES transaction-record-item-id IN format-id
TRBYTES transaction-record-item-id OF subformat-id OF format-id
TRBYTES transaction-record-item-id IN subformat-id IN format-id
```

Returns the size, in bytes, of the referenced data item.

```
TROCCURS transaction-record-item-id OF format-id
TROCCURS transaction-record-item-id IN format-id
TROCCURS transaction-record-item-id OF subformat-id OF format-id
TROCCURS transaction-record-item-id IN subformat-id IN format-id
```

Returns the maximum number of occurrences of the referenced data item. This value is 0 if the argument is not specified as occurring in the TFL source.

```
TRBITS transaction-record-item-id OF format-id
TRBITS transaction-record-item-id IN format-id
TRBITS transaction-record-item-id OF subformat-id OF format-id
TRBITS transaction-record-item-id IN subformat-id IN format-id

TRDIGITS transaction-record-item-id OF format-id
TRDIGITS transaction-record-item-id IN format-id
TRDIGITS transaction-record-item-id OF subformat-id OF format-id
TRDIGITS transaction-record-item-id IN subformat-id IN format-id

TRBYTES transaction-record-item-id OF format-id
TRBYTES transaction-record-item-id IN format-id
TRBYTES transaction-record-item-id OF subformat-id OF format-id
TRBYTES transaction-record-item-id IN subformat-id IN format-id
```

The following table shows how to compute values for TRBITS, TRDIGITS, and TRBYTES. (The value returned is not always the actual size of the item, but rather a value computed from the size.)

Item Type	TRBITS	TRDIGITS	TRBYTES
ALPHA(N)	N*8	N*3	N
BOOLEAN	1	1	1
BOOLEAN (in FIELD)	1	1	1
NUMBER(N)	N*4	N	(N+1) DIV 2
NUMBER(N,M)	N*4	N	(N+1) DIV 2
NUMBER(SN,M)	(N+1)*4	(N+1)	(N+2) DIV 2
REAL	48	12	6
REAL(N)	N*4	N	(N+1) DIV 2
REAL(N,M)	N*4	N	(N+1) DIV 2
REAL(SN,M)	N*4	N	(N+1) DIV 2
FIELD(N)	N	(N+3) DIV 4	(N+7) DIV 8
FIELD(N BOOLEANS)	N	(N+3) DIV 4	(N+7) DIV 8
GROUP	N*4	N	N DIV 2

Transaction Record Variables as Parameters

In transaction processing, most of the work is carried out by Transaction Library procedures. Transaction records are passed to these procedures as parameters.

The formal and actual parameters must refer to the same transaction base, but need not specify the same list of transaction formats. If a procedure is given a transaction record in a format it has not invoked, the procedure is limited as to what it can do with that record.

A transaction record is passed as a parameter in the same way that any 01 entry is passed. For further details, the reader should refer to the COBOL and ANSI74 COBOL reference manuals.

The syntax for a COBOL74 host program to declare transaction records or record arrays as the parameters to a COBOL74 bound procedure is:

```
DECLARATIVES.
section-name SECTION.

          {           { identifier
   USE    { EXTERNAL  { mnemonic-name   AS PROCEDURE
          { --------                    -- ---------  }
          {
          { AS   GLOBAL  PROCEDURE                    }
          { --   ------  ---------                    }
```

For more information, refer to the USE statement as described in the COBOL ANSI-74 Reference Manual.

COBOL74 Specific Constructs for Parameter Passing

The COBOL74 language provides two methods for passing a numeric item to an INTEGER parameter of an ALGOL library entry point. It provides one method for passing a DISPLAY item to a STRING parameter of an ALGOL library entry point. This capability is needed in order to interface COBOL74 programs with the Transaction Library's entry points.

The first way to pass a numeric item to an ALGOL INTEGER parameter is to use the INTEGER function. The INTEGER function will convert an item declared with USAGE COMPUTATIONAL to an ALGOL INTEGER representation.

Another way to pass a numeric item to an ALGOL INTEGER parameter is to specify the BINARY option in the USAGE clause of the item declaration. When BINARY is specified, the item is represented in binary form which is the same as ALGOL INTEGER representation.

COBOL74 provides the STRING function to convert an item declared with USAGE DISPLAY to an ALGOL STRING representation, so that the item can be passed to an ALGOL STRING parameter of a library entry point.

For more information regarding the STRING and INTEGER functions, the reader should refer to the A Series ANSI COBOL74 Reference Manual.

Syntax:

INTEGER (COBOL74 identifier)

STRING (COBOL74 identifier)

Semantics:

INTEGER (COBOL74 identifier)

Converts the COMP item specified by the identifier to an ALGOL INTEGER representation. The identifier must be a COMP item.

STRING (COBOL74 identifier)

Converts the DISPLAY item specified by the identifier to an ALGOL STRING representation so that the item can be passed to an ALGOL STRING parameter of a library entry point.

Examples:

The INTEGER function can be used in both the USING clause and the GIVING clause of the CALL statement,and it will perform the data conversion on both the call to and the return from the library entry point.

```
CALL......USING INTEGER(VARNAME),....
GIVING INTEGER(RESULT).
```

If VARNAME and RESULT are COMP items declared with a USAGE of BINARY, the previous example is changed to:

```
CALL......USING VARNAME,......
GIVING RESULT.
```

The STRING function can be used in the USING clause of the CALL statement. This function performs the data conversion on both the call to and the return from the library entry point.

```
CALL......USING INTEGER(VARNAME),
STRING(DISPLAYVAR) GIVING INTEGER(VARNAME).
```

Transaction Record Variable Assignment

The contents of a transaction record variable can be assigned (that is, copied) to another transaction record variable, provided that both variables represent the same transaction base. This assignment is performed by the MOVE statement. Both the control and data portions of the transaction record are transferred when an assignment is performed.

Syntax:

```
MOVE    transaction-record-1    TO    transaction-record-2
```

Semantics:

transaction-record-1

A fully subscripted transaction record variable whose contents are being assigned or copied to another transaction record variable.

transaction-record-2

A fully subscripted transaction record variable that received its contents from another transaction record variable.

Transaction Use Procedures

COBOL and COBOL74 programs intended for the Update library can declare untyped, parameterless procedures as parameters by naming them in the USING clause of the PROCEDURE DIVISION header and declaring them as a section header in the DECLARATIVES. The CALL statement is used to invoke these procedures. As explained in Section 6 of this manual, the ability to pass procedures as parameters is needed because the user's Update Library must accept two procedures as input. The syntax for the section header of a transaction procedure is:

```
<section name> SECTION. USE AS [GLOBAL] TRANSACTION PROCEDURE.
```

The optional GLOBAL clause allows bound COBOL74 update-library subroutines to reference the transaction procedures received as parameters by the host program. The names of the procedures must be the same in the bound procedure and in the host program.

Examples:

```
PROCEDURE DIVISION USING TR1, TR2.
DECLARATIVES.
TR1 SECTION. USE AS TRANSACTION PROCEDURE.
TR2 SECTION. USE AS TRANSACTION PROCEDURE.
END DECLARATIVES.
PI. CALL TR1.
    CALL TR2.
      .
      .
      .
```

For a bound procedure:

```
DECLARATIVES.
TR1  SECTION.   USE AS GLOBAL TRANSACTION PROCEDURE.
TR2  SECTION.   USE AS GLOBAL TRANSACTION PROCEDURE.
END DECLARATIVES.
```

Transaction Record Use Restrictions

The compiler enforces certain restrictions on the use of transaction record variables. Except as described in the preceding paragraphs, transaction record variables cannot be used without a syntax error. For example, they cannot be used in lists, input/output statements, assignment statements except as described above, or as primaries in expressions.

Compiler Checks

The user language compilers check that all the uses of a particular transaction record variable within a code file are compatible. Parameter checking code ensures that the following attributes of the transaction record variable are those that are expected when the variable is passed as a parameter to a separately compiled code file (the Transaction Library, for example):

1. The transaction record format level.
2. The transaction record control item length.
3. The transaction base creation date-time stamp.

The contents of the variable need not be inspected in order to make this check. If any of these three attribute values do not match, the error message "MISMATCH AT PARAMETER NUMBER <number>, TRANSACTION RECORD <attribute>s DIFFER" is issued when an attempt is made to call a separately compiled code file.

Sample User-Written Program

The following skeleton user-written program in COBOL demonstrates how the transaction base and library entry points are declared. For all COBOL programs represented in this manual, card columns 1 through 6 are not shown; instead, COBOL code ostensibly begins in card column 7. A detailed description of each of the library entry points is given in Section 5 of this manual.

```
IDENTIFICATION DIVISION.
ENVIRONMENT DIVISION.
   .
   .
   .
DATA DIVISION.
FILE SECTION.
   .
   .
   .
DATA-BASE SECTION.
TRANSACTION SECTION.

*  SPECIFY THE TRANSACTION RECORD FORMAT NAMES TO BE USED

TB  TRB = BANKTR
    USING STATIS=STATUS, CREATEACCT, PURGEACCT, DEPOSIT,
          WITHDRAWAL, CHANGEACCT, STATEMENT, TEST,
```

```
          RESTARTDETANKER
          .

*  SPECIFY THE TRANSACTION RECORD VARIABLES

 01  TRIN.
 01  TROUT.
 01  LASTINPUT.
 01  LASTRESPONSE.
 WORKING-STORAGE SECTION.
     .
     .
     .
*****************************************************************
 PROCEDURE DIVISION.
 MAIN SECTION.
     .
     .
     .
      STOP RUN.
*****************************************************************
*
*
 CREATETRUSER.
     CALL "CREATETRUSER        OF BANKTR/CODE/HOSTLIB"
        USING ID-X , IDNUM
        GIVING TPS-RESULT.
*
*
 PURGETRUSER.
     CALL "PURGETRUSER         OF BANKTR/CODE/HOSTLIB"
        USING ID-X
        GIVING TPS-RESULT.
*
*
 LOGONTRUSER.
     CALL "LOGONTRUSER         OF BANKTR/CODE/HOSTLIB"
        USING ID-X, IDNUM
        GIVING TPS-RESULT.
*
*
 LOGOFFTRUSER.
     CALL "LOGOFFTRUSER        OF BANKTR/CODE/HOSTLIB"
        USING IDNUM
        GIVING TPS-RESULT.
*
*
 RETURNRESTARTINFO.
     CALL "RETURNRESTARTINFO   OF BANKTR/CODE/HOSTLIB"
        USING IDNUM, TROUT
        GIVING TPS-RESULT.
*
*
```

```
 RETURNLASTRESPONSE.
     CALL "RETURNLASTRESPONSE   OF BANKTR/CODE/HOSTLIB"
        USING IDNUM, TROUT
        GIVING TPS-RESULT.
*
*
 TANKTRNORESTART.
     CALL "TANKTRNORESTART      OF BANKTR/CODE/HOSTLIB"
        USING IDNUM, TRIN
        GIVING TPS-RESULT.
*
*
 PROCESSTRNORESTART.
     CALL "PROCESSTRNORESTART   OF BANKTR/CODE/HOSTLIB"
        USING IDNUM, TRIN, TROUT
        GIVING TPS-RESULT.
*
*
 OPENTRBASE.
     CALL "OPENTRBASE           OF BANKTR/CODE/HOSTLIB"
        USING OPT, TIMEOUTV
        GIVING TPS-RESULT.
*
*
 CLOSETRBASE.
     CALL "CLOSETRBASE          OF BANKTR/CODE/HOSTLIB"
        GIVING TPS-RESULT.
*
*
 SEEKTRANSACTION.
     CALL "SEEKTRANSACTION      OF BANKTR/CODE/HOSTLIB"
        USING FILENUMBER, BLOCKNUMBER, OFFSETS
        GIVING TPS-RESULT.
*
*
 READTRANSACTION.
     CALL "READTRANSACTION      OF BANKTR/CODE/HOSTLIB"
        USING TRIN
        GIVING TPS-RESULT.
*
*
 SWITCHTRFILE.
     CALL "SWITCHTRFILE         OF BANKTR/CODE/HOSTLIB"
        GIVING TPS-RESULT.
*
*
 HANDLESTATISTICS.
     CALL "HANDLESTATISTICS     OF BANKTR/CODE/HOSTLIB"
        USING STATOPTION
        GIVING TPS-RESULT.
*
*****************************************************************
```

ALGOL INTERFACE TO TPS

The following TPS extensions to the ALGOL language are discussed under separate headings in this section.

1. Invocation of the transaction base.
2. Declaration of transaction record variables.
3. Creation of transaction records.
4. Accessing transaction record items.
5. Transaction record control items.
6. Transaction compile-time functions.
7. Transaction record variables as parameters.
8. Transaction record variable assignment.
9. ALGOL skeleton user-written program.

The TPS application program should not be a library whose entry points invoke the Transaction Library's entry points.

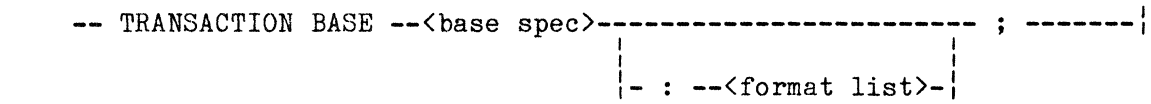
Each program that submits a transaction record for processing must have its own private library for recovery to be successful. If the application program is written as a shared library, then the Transaction Library may not work. The first program that invokes an entry point (which in turn invokes the OPENTRBASE Transaction Library entry point) becomes the only TPS user recognized by the Transaction Library.

Invocation of Transaction Base

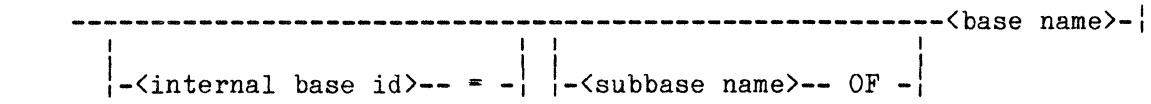
A user program must invoke a transaction base before making any references to formats or items defined within that transaction base. Invocation consists of specifying the transaction base to be invoked, and optionally, specifying a list of transaction record formats and subformats to be invoked. The program may also specify alternate internal names for the transaction base and for any of the formats or subformats invoked. If alternate internal names are specified for the base name, subbase name, format name, or subformat name, then the user-written program must reference these internal identifiers rather than the TFL source identifiers.

A user-written program can also invoke a subbase that has been defined for the transaction base. If a subbase is invoked, then only the transaction record formats and subformats defined within that subbase are accessible to the user-written program. As in transaction base invocation, the program can specify a list of transaction record formats and subformats, possibly using internal names, that can be invoked from that subbase.

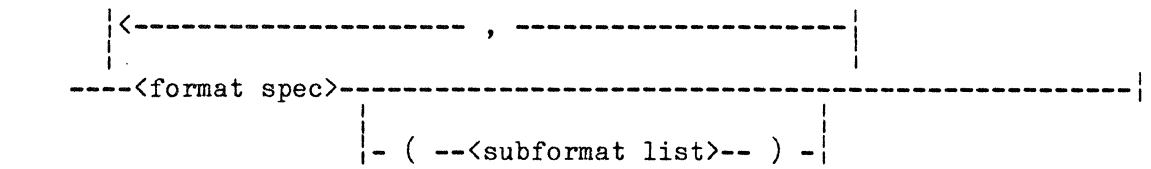
Syntax:



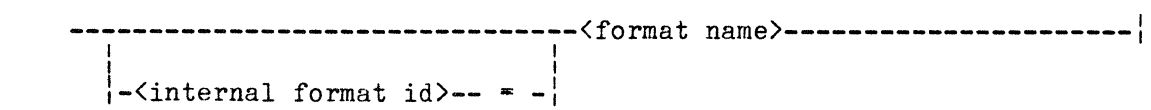
<base spec>



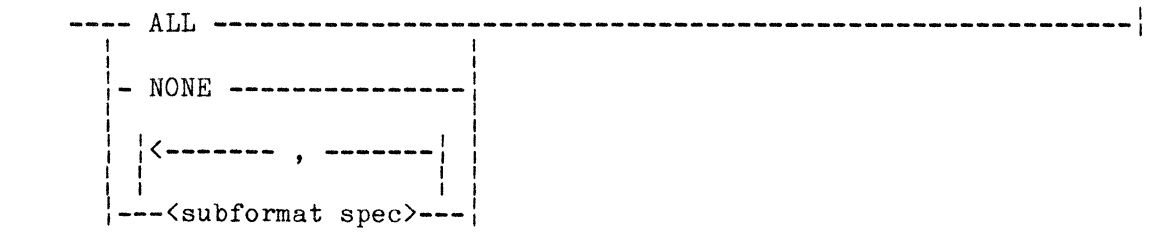
<format list>



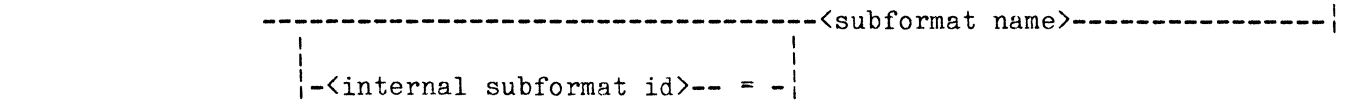
<format spec>



<subformat list>



<subformat spec>



Semantics:

TRANSACTION BASE <base spec>

Specifies the name of the transaction base or transaction subbase to be invoked by the user-written program, and optionally, a list of transaction record formats and subformats. If this list is provided, only the indicated transaction record formats and subformats are invoked. If this list is not provided, all transaction record formats and subformats are invoked for the transaction base or subbase that has been invoked.

: <format list>

A list of transaction record format and subformat names, possibly internal names, that are invoked by the user-written program. If only a <format name> is specified in the <format spec>, then by default all subformats of that format are invoked.

<internal base id>=<base name>
<subbase name> OF <base name>

Specifies the internal name of the transaction base being invoked, or the name of the transaction subbase being invoked.

<format spec>,
<format spec> (<subformat list>),
<format name>
<internal format id>=<format name>

Specifies the name of the transaction record format being invoked, and optionally, its internal name, an indication of the transaction record subformats to be invoked, or both.

<subformat list>

Indicates the specific subformats of a transaction record format that the program invokes. If no <subformat list> is provided for a particular <format name>, then ALL is assumed.

ALL

If a transaction base with a list of formats has been invoked, specifying ALL invokes all of the subformats of that format. If a transaction subbase has been invoked, specifying ALL invokes only those subformats specified for this format in the TFL subbase declaration.

If only a format name is listed in the TFL subbase declaration, then by default TFL includes all subformats of the format in the subbase declaration.

NONE

If NONE is specified for a particular transaction format, then no subformats are invoked.

<subformat spec>

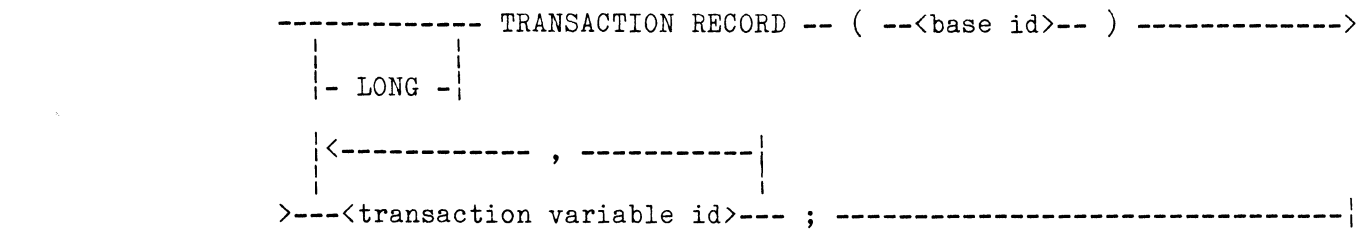
If a list of <subformat spec>s is specified, only those subformats on the list are invoked. If a transaction subbase is invoked, then <subformat spec>s can include only those subformats defined within the transaction subbase for a particular format.

Declaration of Transaction Record Variables

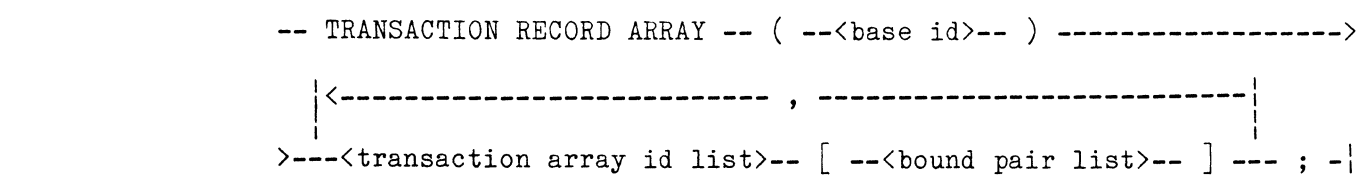
A transaction record is an array row that can contain the transaction data of one of several transaction formats declared in the TFL source. A transaction record variable names one of these array rows. By allowing a transaction record variable to contain the transaction data of one of several transaction formats, the record becomes, in effect, a structured variable. A transaction record variable can only be associated with one transaction base or transaction subbase and can contain only formats and subformats that have been invoked from that particular transaction base or transaction subbase. The size of the array row is large enough to accommodate the largest of all the formats invoked for it.

Syntax:

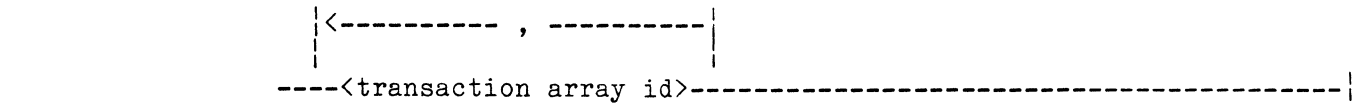
<transaction record declaration>



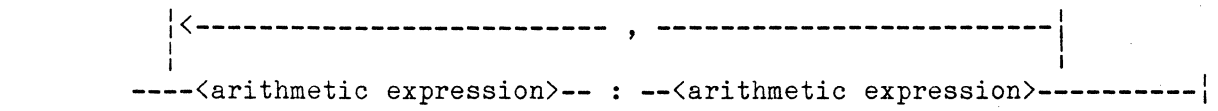
<transaction record array declaration>



<transaction array id list>



<bound pair list>



Semantics:

<arithmetic expression>

An arithmetic expression as defined in the ALGOL Reference Manual.

<transaction variable id>

An identifier that is the name of a transaction record variable.

<transaction array id>

An identifier that is the name of an array of transaction record variables. Each fully subscripted element of a <transaction array id> is a transaction record.

<base id>

The name, or internal name, of a transaction base or transaction subbase invoked in the program. Specifying a <base id> in the declaration of a transaction record or transaction record array associates a transaction base or transaction subbase with the particular record(s).

LONG

Suppresses the segmentation of transaction records. Ordinarily, transaction records larger than 1024 words are segmented into 512-word entities. (This segmentation is standard for all ALGOL arrays declared to have more than 1024 elements.)

<bound pair list>

Gives the lower and upper bounds of all subscripts taken in order from left to right. For more information, refer to Section 4 in the ALGOL Reference Manual.

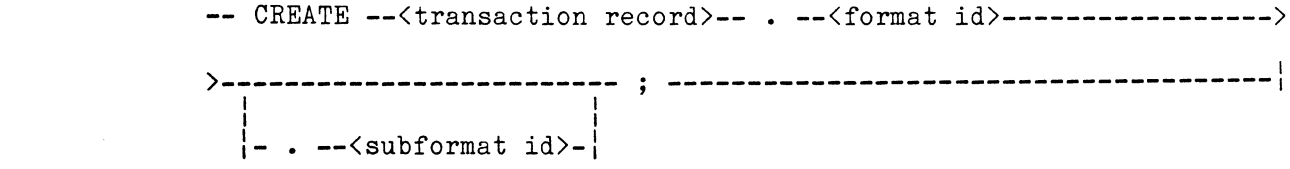
Creation of Transaction Records

The contents of a transaction record variable are undefined until the variable is initialized to a particular format by a CREATE statement. CREATE assigns the initial values of all items in the transaction record format (and transaction subformat, if given) to the record variable and initializes the transaction record control items. When a format is created, only those items in the common part are assigned initial values. When a subformat is created, the common part items as well as the subformat part items are assigned initial values. The record variable continues to contain the given format until it is reinitialized by a subsequent CREATE. It is never cleared by the system.

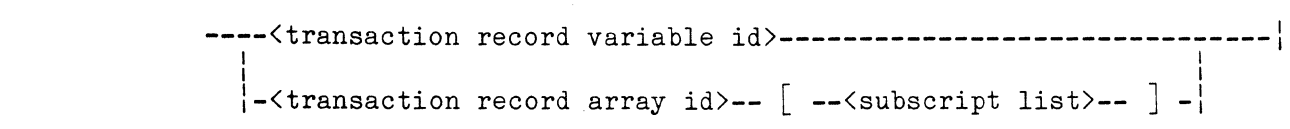
Once a transaction record variable has been created in a particular transaction format and, optionally, subformat, the items defined within the format and subformat can be accessed and manipulated. If a transaction record is created in a particular transaction record format, the record only contains the data items associated with that transaction record format. If a transaction record is created in a particular transaction record format and subformat, then the record contains the data items associated with the format and the data items associated with the subformat.

Syntax:

<create statement>



<transaction record>



<subscript list>

Semantics:

<transaction record>

The name of the transaction record variable that is initialized. If a transaction record array element is referenced, it must be fully subscripted.

<format id>
<subformat id>

The names of the format and subformat (if given) whose data item's initial values are assigned to the record variable.

<subscript list>

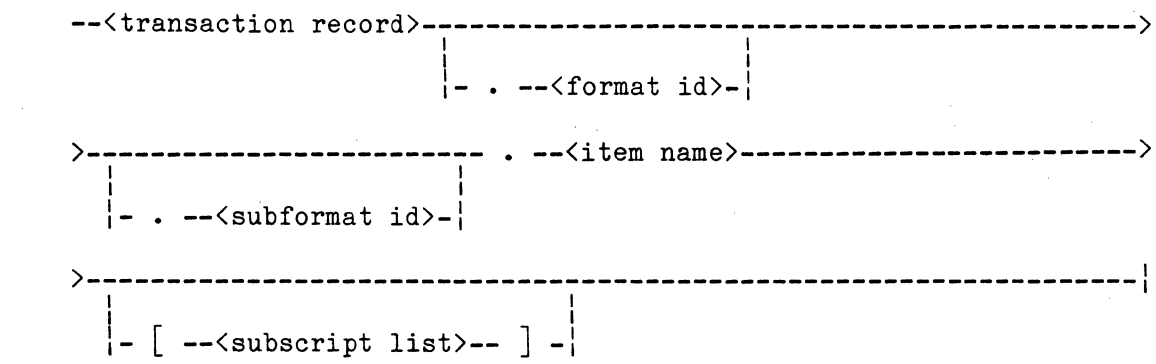
A list of one or more <subscript>s that are required to qualify the referenced item. In this case, a <transaction array id>. <subscript> is defined as any legitimate ALGOL <arithmetic expression>. The reader should refer to the ALGOL Reference Manual for further details.

Accessing Transaction Record Items

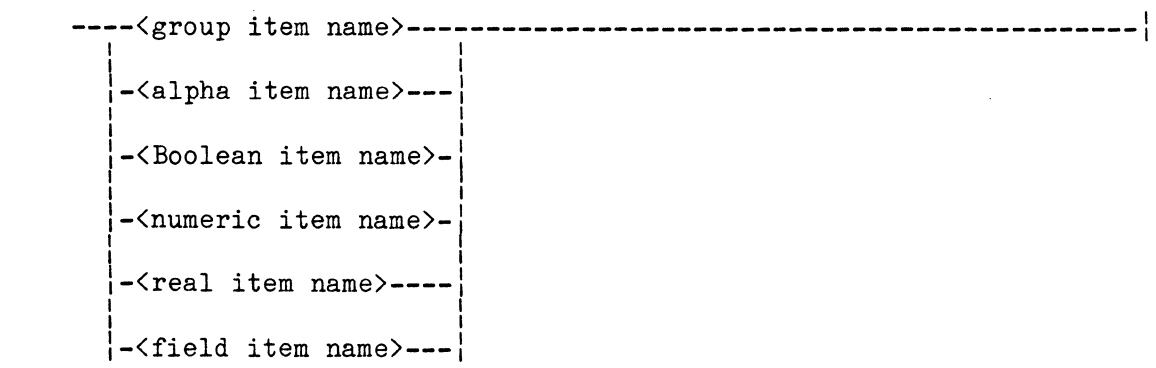
A transaction record can contain a transaction of any format and subformat that is declared for it in the TFL source. The user can reference data items in the declared format and subformat of that transaction. Transaction record data items are considered normal data items and can be referenced at any place that is acceptable for a normal data item.

Syntax:

<item reference>



<item name>



Semantics:

<transaction record>

The name of a transaction record variable. If a transaction record array element is used, it must be fully subscripted.

<format id>

This construct is optional,unless it is required for qualification. Refer to "Requirements For Data Item Qualification" in this section for more information.

<subformat id>

This construct is optional, unless it is required for qualification. Refer to "Requirements For Data Item Qualification" in this section for more information.

<item name>

An item within the transaction record format or subformat presently occupying the record variable. <item name> must be fully subscripted if it is an element of an occurring item.

<item name> can be used either as the left part of an assignment or REPLACE statement or as a primary in an expression. The type of the item must be consistent with the context in which it is used. Refer to "Appearance Of TFL Items To Application Languages" in Section 3 of this manual for more information.

Data items of transaction record formats or subformats that are occurring items, items embedded within one or more occurring groups, or items that are occurring and embedded within occurring groups must be subscripted. The <subscript>s within a <subscript list> are listed from left to right, from the outer-most occurring GROUP to the inner-most occurring GROUP or occurring items. Subscripting is illustrated by the following example, using a TFL fragment of a transaction record format. The fourth item of E, in the second GROUP of C, in the seventh GROUP of A would be referenced with the subscript triple (7,2,4). Referencing data item D in the eighth GROUP of C, and the first GROUP of A would require the subscript pair (1,8).

```
 A    GROUP
(
   B  REAL;
   C  GROUP
   (
     D REAL;
     E NUMBER(6) OCCURS 5 TIMES;
   ) OCCURS 10 TIMES;
   ) OCCURS 20 TIMES;
```

Requirements For Data Item Qualification

The amount of qualification required to access a data item of a particular transaction record format or subformat varies. In every case, however, the transaction record variable containing the desired data item must be referenced. The following cases show the varying requirements for qualification:

Case 1. Data Item Qualification

If the name of the desired data item is unique with respect to data items of other invoked formats, then only the data item name need be specified.

Example:

```
TRANREC.DATAITEMNAME
```

Case 2. Format Name and Data Item Name Qualification

If the name of the desired data item is not unique with respect to data items of other invoked formats, but is unique to the format that contains it, then both the format and the data item name are needed for qualification.

Example:

```
TRANREC.FORMATNAME.DATAITEMNAME
```

Case 3. Subformat Name and Data Item Name Qualification

Both the subformat name and the data item name are needed for qualification when the following conditions prevail:

1. The name of the desired data item is not unique with respect to the common portion of another invoked format.
2. The name of the desired data item is contained within a subformat.

3. Another data item within a different subformat of the same format has the same name as the desired data item.

Also, if the desired data item is contained within a subformat whose name is unique to all invoked formats and subformats, and the desired data item is not unique with respect to a subformat of another format, then both the subformat name and the data item name are needed for qualification.

Example:

```
TRANREC.SUBFORMATNAME.DATAITEMNAME
```

Case 4. Format Name, Subformat Name, and Data Item Name Qualification

The format name, subformat name, and data item name are all required for qualification if the following statements are all true:

1. The desired item is not unique with respect to a subformat of another invoked format.

2. The item is not unique with respect to the format that contains it.

3. The name of the subformat that contains the desired item is not unique with respect to all invoked formats and subformats.

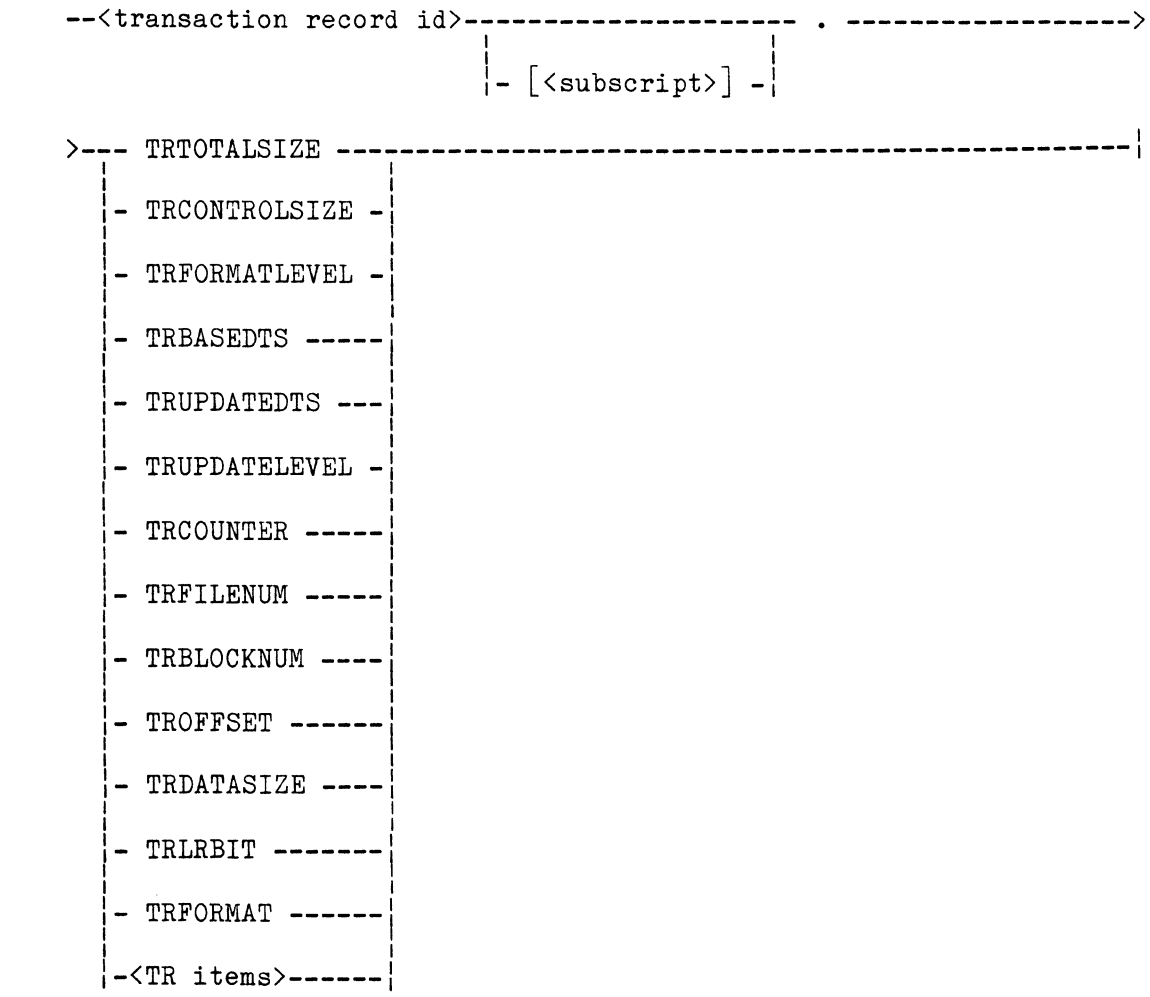
Example:

```
TRANREC.FORMATNAME.SUBFORMATNAME.DATAITEMNAME
```

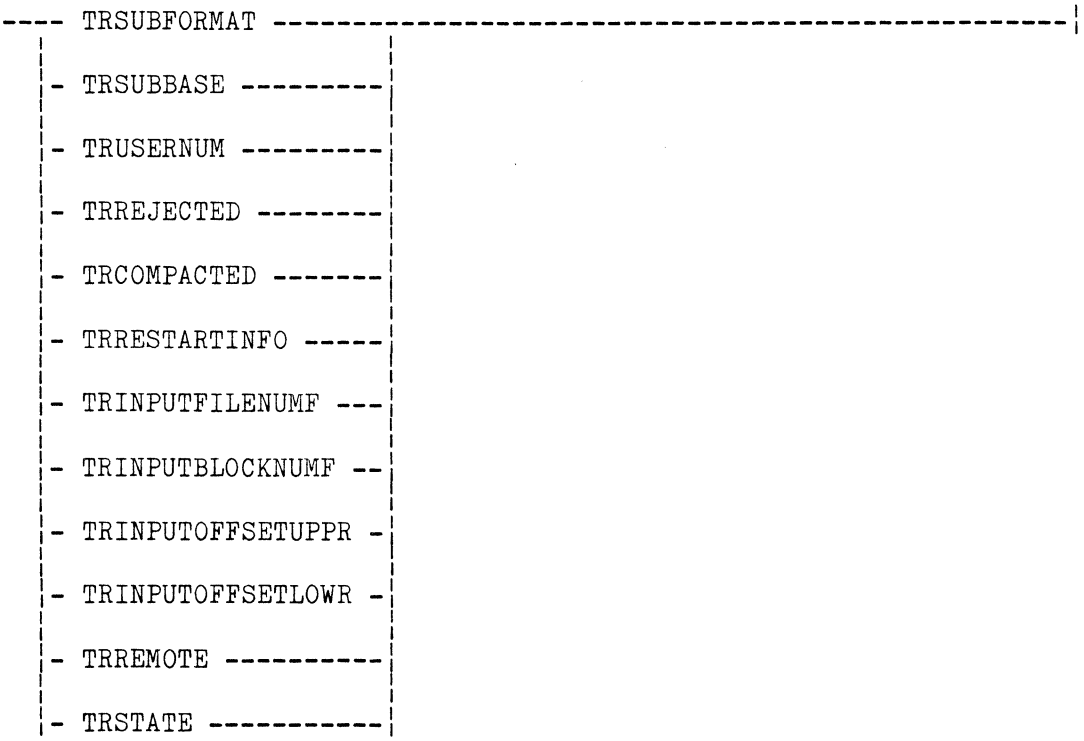
Transaction Record Control Items

Control items are system-defined items contained within every transaction record. These items are maintained by the Transaction Processing System and are read-only in all BDMSALGOL programs. The initial values of these control items are assigned when a transaction record is created. These items are defined only after a transaction record has been CREATEd with a particular format and, possibly, subformat.

Syntax:



<TR items>



Semantics:

<transaction record id>

A transaction record variable. This variable must be fully subscripted if a transaction record array element is used.

TRTOTALSIZE

A numeric item that specifies the total size, in bytes, of the transaction record, including control and data parts. The size of the control part is a constant for all records.

TRCONTROLSIZE

A numeric item that specifies the size, in bytes, of the control portion of the transaction record. This size is fixed for all transaction records.

TRFORMATLEVEL

A numeric item that specifies the format level of the transaction record layout, defined as 1.

TRBASEDTS

A numeric item that contains the creation date-time stamp (with the TIME(6) value as defined in the ALGOL Reference Manual) associated with the TRDESCRIPTION file. TRBASEDTS is used to coordinate the TRDESCRIPTION file, user programs, and the Transaction Library.

TRUPDATEDTS

A numeric item that specifies the update date-time stamp that defines the update level relative to the version of the TRDESCRIPTION used to compile a program. TRUPDATEDTS is also used to coordinate the various code files.

TRUPDATELEVEL

A numeric item that specifies the TFL update level, which is used to help coordinate TPS code files and user program code files. TRUPDATELEVEL is used when comparing transaction base update time stamps.

TRCOUNTER

A numeric item that contains a unique numeric value that synchronizes input and response transaction records between a remote system and the host system.

TRFILENUM
TRBLOCKNUM
TROFFSET

TRFILENUM, TRBLOCKNUM, and TROFFSET are all numeric items. Each refers to the record address of a transaction record in a journal data file. Each is assigned by the Transaction Library when the transaction is first written to a transaction journal data file.

TRFILENUM gives the number of the journal data file. TRBLOCKNUM gives the block number (in other words, the beginning segment address) within that file, and TROFFSET gives the word offset within that block.

TRDATASIZE

A numeric item that specifies the length, in bytes, of data items in a transaction record.

TRLRBIT

A bit used to distinguish a left control word from a right control word. TRLRBIT is defined as TRUE for a left control word; it is a numeric item.

TRFORMAT

A numeric item that contains a unique numeric value that identifies the transaction record format contained within a transaction record.

TRSUBFORMAT

A numeric item that contains a unique numeric value that identifies the transaction subformat contained within a transaction record. The value of TRSUBFORMAT is unique for a given TRFORMAT value.

TRSUBBASE

A numeric item that contains a unique numeric value that identifies the transaction subbase that has been invoked.

TRUSERNUM

A numeric item that identifies the source or owner of a transaction record, primarily to aid in recovery and restart after an interruption in processing. The last good input record and its response record are saved for each value of TRUSERNUM.

TRREJECTED

A Boolean item that is used by the Transaction Library to mark a transaction record that has already been written to the history file, and is subsequently rejected by the system when faults occur during processing.

TRCOMPACTED

A Boolean item that is used by the Transaction Library to mark a transaction record that is in its compacted form. A transaction record that cannot be compacted has a value of 0. The TRCOMPACTED field may hold the following values:

Value	Meaning
0	Transaction record cannot be compacted.
2	Transaction record can be compacted, but is not in compacted form.
3	Transaction record can be compacted and is in compacted form.

TRRESTARTINFO

A Boolean item that is used by the Transaction Library to designate how the transaction record that is contained within a transaction journal is written to the journal. The transaction record is written in one of the following ways, depending on the TRRESTARTINFO value:

Value	Meaning
0	No restart record follows this input transaction record.
2	A restart record follows this input transaction record, where the transaction record is the input record of the input-restart record pair.
1	A restart record follows this input transaction record, where the transaction record is the restart record of input-restart record pair.

TRINPUTFILENUMF
TRINPUTBLOCKNUMF
TRINPUTOFFSETUPPR
TRINPUTOFFSETLOWR

These control items are reserved for system use and refer to the addresses of input records in the transaction journal data file. Each of these control items represents a numeric value.

TRREMOTE

TRREMOTE is a Boolean item. It is a field that is used to distinguish between a record submitted from a remote system and a record submitted from the host system. If a transaction record is submitted from a remote system, TRREMOTE is defined to be TRUE.

TRSTATE

A numeric item that specifies the state of a transaction record during its processing period. TRSTATE is used to check whether a transaction record has passed through the appropriate states during its processing period.

subscript

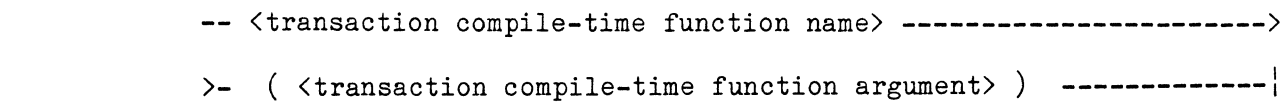
An ALGOL <arithmetic expression> that identifies a particular transaction record variable within an array of transaction record variables.

Transaction Compile-Time Functions

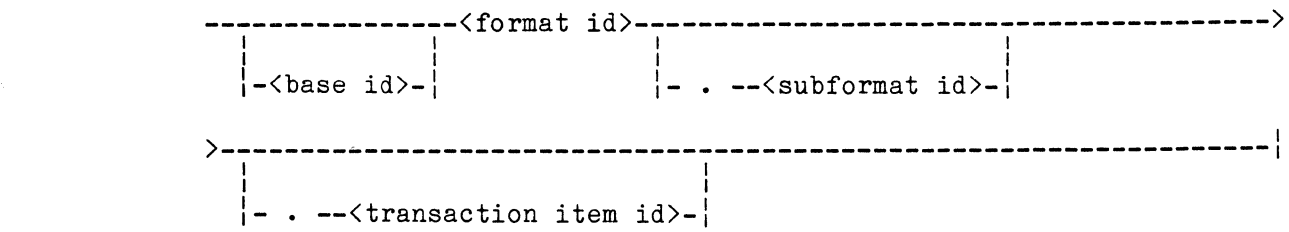
Transaction compile-time functions provide access to certain properties of transaction record formats that are constant at compile time. These compile-time constructs are particularly useful when coding an Update Library. For examples of the use of compile-time functions, the reader should refer to Section 6 of this manual.

Syntax:

<transaction compile-time functions>



<transaction compile-time function argument>



Semantics:

<transaction item id>

The name of a data item contained within an invoked transaction format or subformat.

<subformat id>

The name of a transaction subformat that has been invoked within the program.

<format id>

The name of a transaction format that has been invoked within the program.

<base id>

The name of a transaction base that has been invoked within the program.

The following list specifies which transaction compile-time functions apply to the possible arguments, and what each function represents. Not all functions apply to all arguments. <base id> needs to be referenced only when transaction base qualification is required.

TRFORMAT <format id>

Returns the numeric value assigned to the <format id>.

TRSUBFORMAT <format id>.<subformat id>

Returns the numeric value assigned to the <subformat id>.

TRDATASIZE <format id>

Returns the size in bytes of the common part of the transaction record.

TRDATASIZE <format id>.<subformat id>

Returns the size, in bytes, of the subformat record (including the common part).

TRBITS <format id>.<transaction item id>
TRBITS <format id>.<subformat id>.<transaction item id>

Returns the size, in bits, of the referenced data item.

TRDIGITS <format id>.<transaction item id>
TRDIGITS <format id>.<subformat id>.<transaction item id>

Returns the size, in digits, of the referenced data item.

TRBYTES <format id>.<transaction item id>
TRBYTES <format id>.<subformat id>.<transaction item id>

Returns the size, in bytes, of the referenced data item.

TROCCURS <format id>.<transaction item id>
TROCCURS <format id>.<subformat id>.<transaction item id>

Returns the maximum number of occurrences of the referenced data item. This value is 0 if the argument is not specified as occurring in the TFL source.

TRBITS <format id>.<transaction item id>
TRBITS <format id>.<subformat id>.<transaction item id>

TRDIGITS <format id>.<transaction item id>
TRDIGITS <format id>.<subformat id>.<transaction item id>

TRBYTES <format id>.<transaction item id>
TRBYTES <format id>.<subformat id>.<transaction item id>

The following table shows how to compute values for TRBITS, TRDIGITS, and TRBYTES. (The value returned is not always the actual size of the item, but rather a value computed from the size.)

Item Type	TRBITS	TRDIGITS	TRBYTES
ALPHA(N)	N*8	N*3	N
BOOLEAN	1	1	1
BOOLEAN (in FIELD)	1	1	1
NUMBER(N)	N*4	N	(N+1) DIV 2
NUMBER(N,M)	N*4	N	(N+1) DIV 2
NUMBER(SN,M)	(N+1)*4	N+1	(N+2) DIV 2
REAL	48	12	6
REAL(N)	N*4	N	(N+1) DIV 2
REAL(N,M)	N*4	N	(N+1) DIV 2
REAL(SN,M)	N*4	N	(N+1) DIV 2
FIELD(N)	N	(N+3) DIV 4	(N+7) DIV 8
FIELD(N BOOLEANS)	N	(N+3) DIV 4	(N+7) DIV 8
GROUP	N*4	N	N DIV 2

Transaction Record Variables as Parameters

In transaction processing, most of the work is carried out by Transaction Library procedures. Transaction records are passed to these procedures as parameters.

The formal and actual parameters must be invoked from the same transaction base, but they need not specify the same list of transaction record formats. If a transaction record is passed to a procedure in a format that the record has not invoked, then the procedure is limited as to what it can do with that record.

Transaction records can not be passed to intrinsics or to external procedures initiated through a CALL or PROCESS statement.

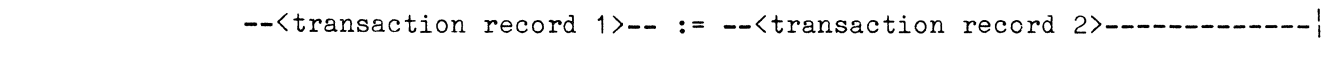
The specification syntax for a formal transaction record variable is identical to the syntax given in this section under Declaration of Transaction Record Variables, except that the <bound pair list> is replaced by a <lower bound list> as it is replaced for any arrays. (The transaction base must have been invoked before specifying the syntax for a formal transaction record variable.) For further information regarding the semantics of <lower bound list>, the reader should refer to section 4 in the ALGOL Reference Manual.

Transaction Record Variable Assignment

The contents of a transaction record variable can be assigned (that is, copied) to another transaction record variable, provided that both variables represent the same transaction base. Both the control and data portions of the transaction record are transferred when an assignment is performed.

In DMALGOL, the implementation language for the Transaction Library, an ARRAY reference variable can be assigned to a transaction record variable. This construct is not permitted in user-written programs.

Syntax:



Semantics:

transaction-record-1

A transaction record variable that received its contents from another transaction record variable.

transaction-record-2

A transaction record variable whose contents are being assigned or copied to another transaction record variable.

Transaction Record Use Restrictions

The compiler enforces certain restrictions on the use of transaction record variables. Except as described in the preceding paragraphs, transaction record variables cannot be used without a syntax error. For example, they cannot be used in lists, input/output statements, assignment statements except as described above, or as primaries in expressions.

Compiler Checks

The user language compilers check that all the uses of a particular transaction record variable within a code file are compatible. Parameter checking code ensures that the following attributes of the transaction record variable are those that are expected when the variable is passed as a parameter to a separately compiled code file (the Transaction Library, for example):

1. The transaction record format level.
2. The transaction record control item length.
3. The transaction base creation date-time stamp.

The contents of the variable need not be inspected in order to make this check. If any of these three attribute values do not match, the error message "MISMATCH AT PARAMETER NUMBER <number>, TRANSACTION RECORD <attributes>s DIFFER" is issued when an attempt is made to call a separately compiled code file.

Sample User-Written Program

A skeleton user-written program in ALGOL is shown below to demonstrate how the transaction base and Transaction Library entry points are declared in each language. A detailed description of the library entry points is given in Section 5 of this manual.

```
BEGIN   % SAMPLE BATCH PROGRAM USING TRANSACTIONS

    TRANSACTION BASE TRB = BANKTR;
    LIBRARY L(TITLE="BANKTR/CODE/HOSTLIB.");

% DECLARE ALL ENTRYPOINTS TO BE USED


INTEGER PROCEDURE CREATETRUSER(IDSTRING, IDNUM);
    STRING IDSTRING; INTEGER IDNUM;
```

```
    LIBRARY L;
INTEGER PROCEDURE PURGETRUSER(IDNUM);
    INTEGER IDNUM;
    LIBRARY L;
INTEGER PROCEDURE LOGONTRUSER(IDSTRING, IDNUM);
    STRING IDSTRING; INTEGER IDNUM;
    LIBRARY L;
INTEGER PROCEDURE LOGOFFTRUSER(IDNUM);
    INTEGER IDNUM;
    LIBRARY L;
INTEGER PROCEDURE RETURNRESTARTINFO(IDNUM, TROUT);
    INTEGER IDNUM;
    TRANSACTION RECORD (TRB) TROUT;
    LIBRARY L;
INTEGER PROCEDURE RETURNLASTRESPONSE(IDNUM, TROUT);
    INTEGER IDNUM;
    TRANSACTION RECORD (TRB) TROUT;
    LIBRARY L;
INTEGER PROCEDURE TANKTRNORESTART(IDNUM, TRIN);
    INTEGER IDNUM;
    TRANSACTION RECORD (TRB) TRIN;
    LIBRARY L;
INTEGER PROCEDURE PROCESSTRNORESTART(IDNUM, TRIN, TROUT);
    INTEGER IDNUM;
    TRANSACTION RECORD (TRB) TRIN, TROUT;
    LIBRARY L;
INTEGER PROCEDURE OPENTRBASE(USEROPTION, TIMEOUT);
    INTEGER USEROPTION, TIMEOUT;
    LIBRARY L;
INTEGER PROCEDURE CLOSETRBASE;
    LIBRARY L;
INTEGER PROCEDURE SEEKTRANSACTION(FILENUM, BLOCKNUM, OFFSET);
    INTEGER FILENUM, BLOCKNUM, OFFSET;
    LIBRARY L;
INTEGER PROCEDURE READTRANSACTION (TRREC);
    TRANSACTION RECORD (TRB) TRREC;
    LIBRARY L;
INTEGER PROCEDURE SWITCHTRFILE;
    LIBRARY L;
INTEGER PROCEDURE HANDLESTATISTICS(STATOPTION);
    VALUE STATOPTION;
    INTEGER STATOPTION;
    LIBRARY L;

 % DECLARE TRANSACTION RECORD VARIABLES TO BE USED

    TRANSACTION RECORD (TRB)
         TRIN,
         TROUT,
         LASTINPUT,
         LASTRESPONSE;
```

```
    STRING JOURNALNAME;
      .
      .
      .
      .
%   START OF PROGRAM
%   SET LIBPARAMETER IN DECLARATION OR BEFORE FIRST CALL ON ENTRYPOINT

    L.LIBPARAMETER := JOURNALNAME;

%   BODY OF PROGRAM

      .
      .
      .
END.
```

5. TRANSACTION LIBRARY

INTRODUCTION

The Transaction Library is a collection of procedures that are accessed by user-written programs to process or tank transactions and read them back from transaction journal files. The procedures are accessed through a set of entry points supplied by the Transaction Library, which is tailored for a particular transaction base during compilation. The Transaction Library performs such functions as calling the Update Library to process a transaction against a data base, saving transaction records in transaction journal files, and automatically reprocessing transactions backed out by DMSII Recovery.

In a shared resources system, the Transaction Library must reside on the host system (the system that contains the data base), where it is known as the "Host Library." A second version of the Transaction Library, known as the "Remote Library," resides on any remote system that interfaces with the host system. The Remote Library, described in Section 7 of this manual, appears identical to the Host Library from the user-written program's viewpoint. The Remote Library communicates with the host system, where most processing is actually performed.

COMPILING THE TRANSACTION LIBRARY

Transaction Library compilation can be initiated automatically by SYSTEM/TFL whenever a new TRDESCRIPTION file is created. Automatic compilation occurs if the TFL ZIP dollar-card option is set. Refer to "SYSTEM/TFL Dollar Card Options" in Section 3 for a description of this option.) The title of the resulting Transaction Library code file is <base name>/CODE/HOSTLIB.

Alternatively, the user can compile the Transaction Library by compiling the source files TRBASE/HOSTLIB, the TRDESCRIPTION file, and portions of TRBASE/PROPERTIES and TRBASE/UTILITY with the DMALGOL compiler, as shown in Figure 5-1. All required processor options are included in the TRBASE/HOSTLIB source file. The user must label-equate the internal DMALGOL compiler file named DASDL to the appropriate TRDESCRIPTION file named <transaction base name>/TRDESCRIPTION.

A compile-time check ensures that the TRBASE/PROPERTIES property level matches the property level contained within the TRDESCRIPTION file. If the level of the TRBASE/PROPERTIES file exceeds the level of the TRDESCRIPTION file, the syntax error "DESCRIPTION FILE TOO OLD FOR PROPERTIES" is issued. If the description file level exceeds the TRBASE/PROPERTIES level, the syntax error "PROPERTIES TOO OLD FOR DESCRIPTION FILE" is issued.

Example:

The following example shows a WFL job for compiling a Transaction Library:

```
?JOB COMPILEHOSTLIB;
USER = SITE;
FAMILY = TPS OTHERWISE DMS;
COMPILE BANKTR/CODE/HOSTLIB WITH DMALGOL(CT) LIBRARY;
SUBSYSTEM = BLUE;
OPTION = FAULT DSED ARRAYS;
PL SUBSPACES = 0;
PL FILE DASDL = BANKTR/TRDESCRIPTION;
PL FILE TAPE = TRBASE/HOSTLIB;
PL FILE TRUTILITY = TRBASE/UTILITY;
PL FILE TRPROPERTIES = TRBASE/PROPERTIES;
PL FILE ERRORFILE(KIND=DISK,PROTECTION=SAVE,TITLE=ERROR/HL);
DATA CARD;
$MERGE LINEINFOERRLIST;
IF CT ISNT COMPILEDOK THEN
    ABORT "*** SYNTAX ERROR ***";
?END JOB;
```

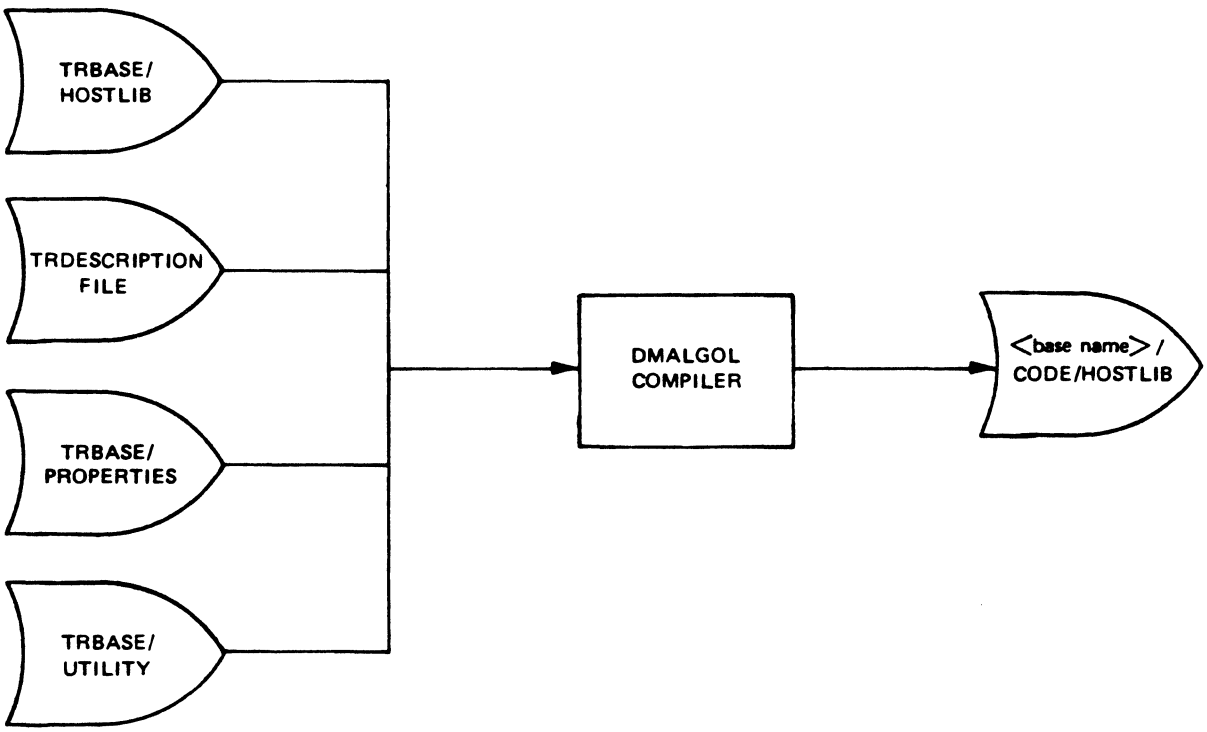


Figure 5-1. Compilation of the Transaction Library

LIBRARY INITIATION

As with any library, a user-written program's first attempt to call a library entry point causes the MCP to connect the program to the library stack. Physically, the Transaction Library consists of the following stacks:

1. A D2 (and D1) stack, which is shared by all users of a transaction base. The D2 stack is also known as the <transaction base name>/CODE/HOSTLIB stack.

2. A D3 stack for each active transaction journal for that transaction base. These stacks are also known as the <transaction journal name>/SHARED/LIBRARY stacks. Each D3 stack is shared by all users of that journal.

3. A private D4 stack for each user-written program's Transaction Library invocation. This stack is also known as the <transaction journal name>/PRIVATE/LIBRARY stack.

The first call on a Transaction Library entry point causes the private D4 stack and any of the required shared stacks that are not already active to be initiated.

In general, a user-written program calls Transaction Library entry points that reside in the D4 stack. The procedures in the the D4 stack can, in turn, call procedures in the D3 and D2 stacks. Data and procedures that are global to a particular transaction base reside in the D2 stack. Data and procedures that are global to a particular transaction journal reside in the D3 stack. Data and procedures that are local to a Transaction Library invocation reside in the D4 stack.

The user-written program is responsible for specifying which transaction journal it wishes to process against. This specification determines the shared journal library (in other words, the D3 library stack) that the program accesses. For transaction processing against a data base, the specified journal must be the TRHISTORY journal. For transaction tanking, the specified journal can be any journal except the TRHISTORY journal. For transaction reading, any journal can be specified.

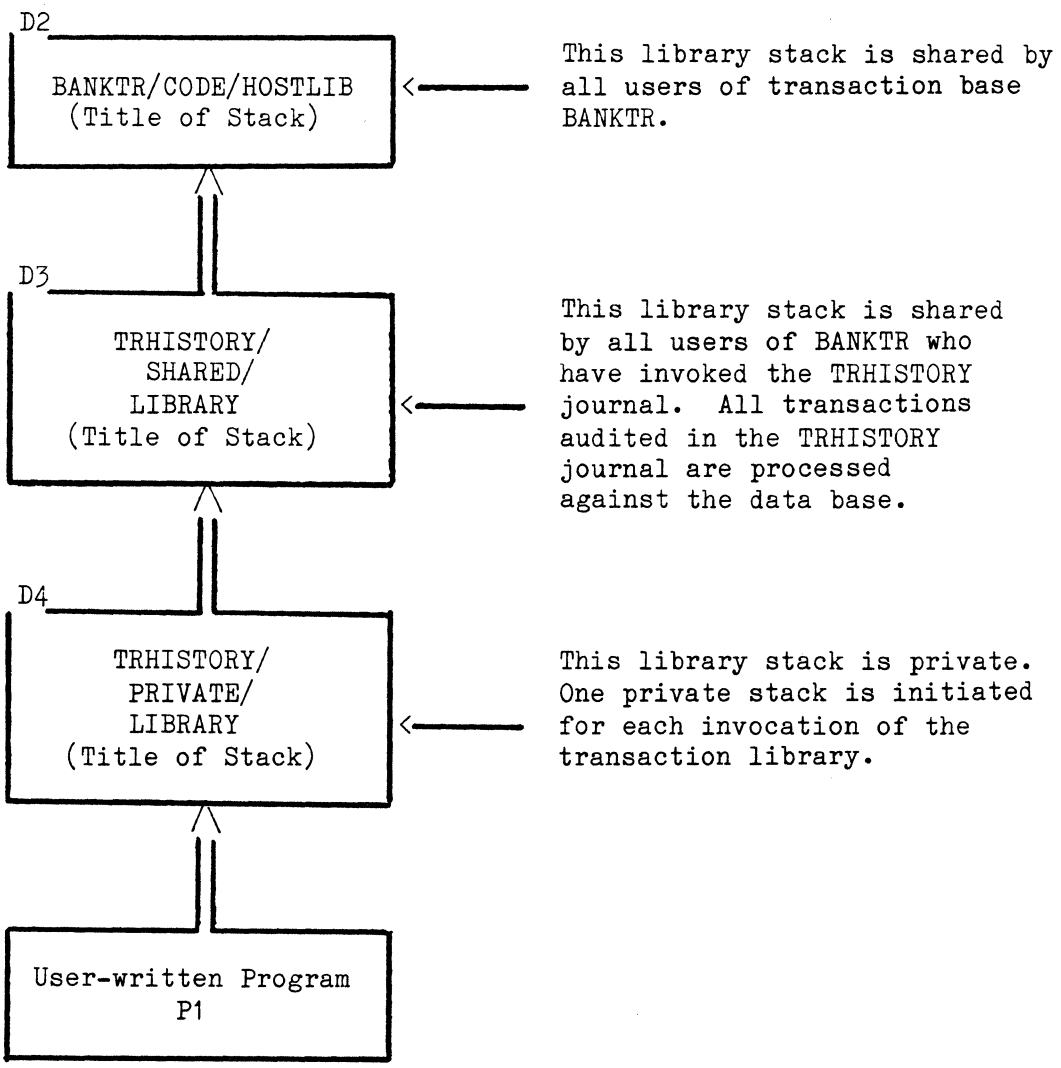
A journal name is specified by setting the LIBPARAMETER library attribute equal to the desired journal name. The LIBPARAMETER attribute must be set before the first call on a Host Library entry point. (For details regarding the LIBPARAMETER library attribute, the reader should refer to the A Series System Software Utilities Reference Manual.)

For a single transaction base, user programs can access a maximum of 128 journals at any one time. Similarly, a maximum of 128 private library stacks can be active for a single journal at any one time.

Examples:

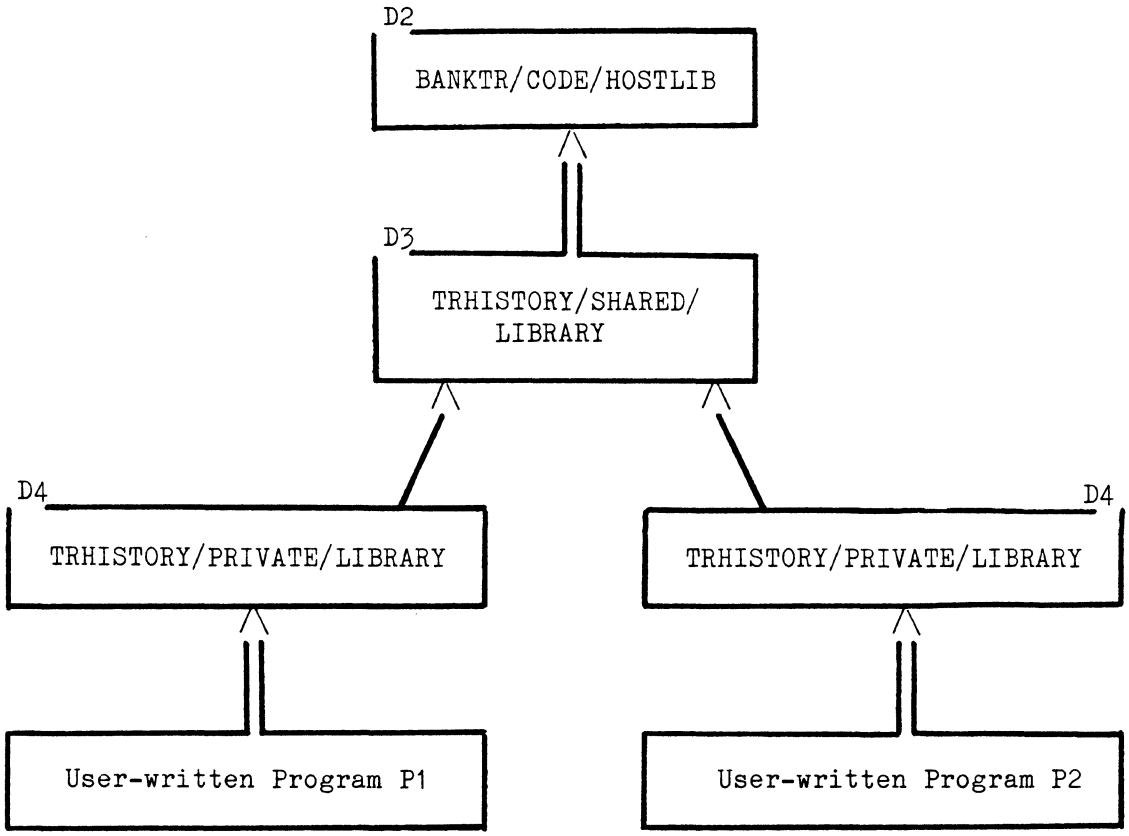
The dynamic nature of the Transaction Library is illustrated in the following examples, which assume the existence of a transaction base named BANKTR.

A user-written program named P1 is initiated to process transactions, which have been defined in transaction base BANKTR, against a data base. No other programs access BANKTR at this time. After P1's first call on a Transaction Library entry point, the following diagram illustrates the stack structure that would result. The arrows in the diagram indicate which stacks can call procedures in which other stacks. The user-written Update Library module is not shown in this diagram.

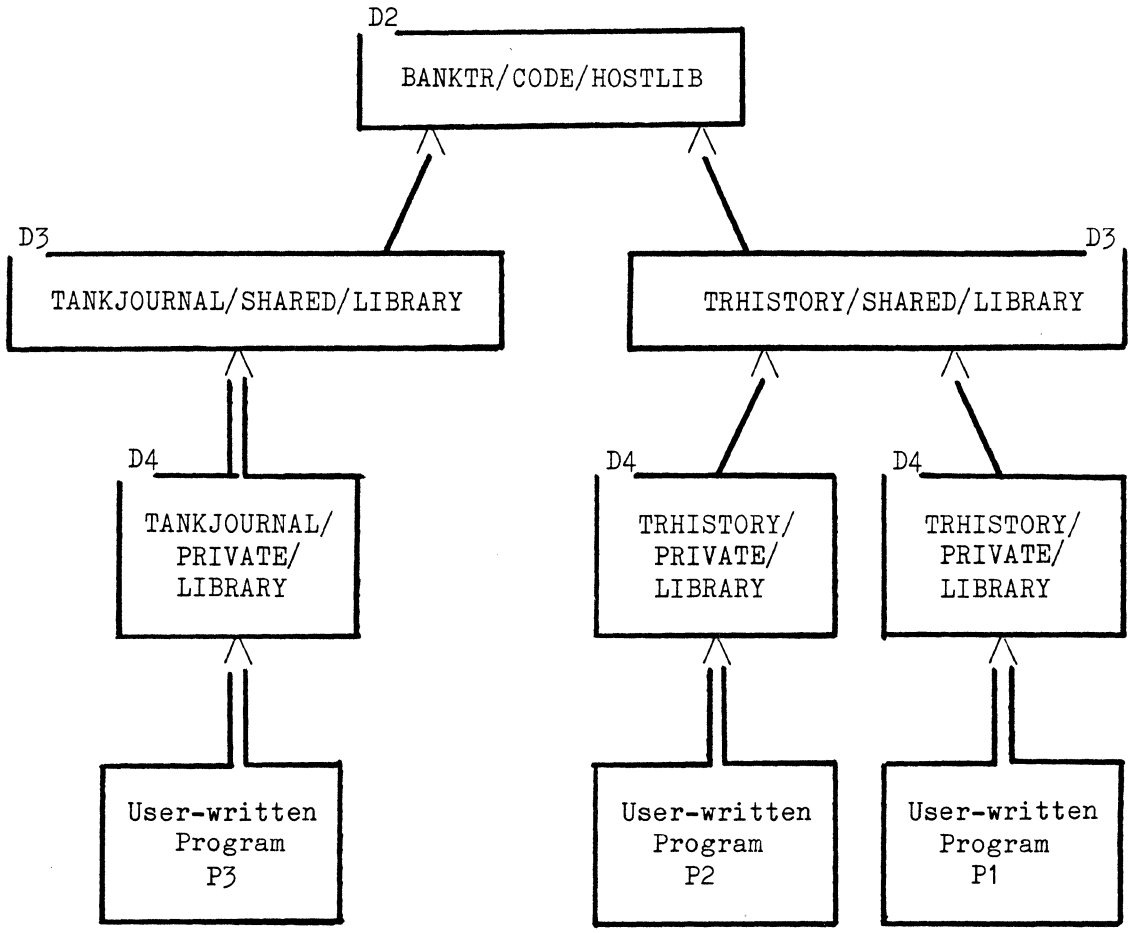

D2
BANKTR/CODE/HOSTLIB
(Title of Stack)
This library stack is shared by all users of transaction base BANKTR.
D3
TRHISTORY/
SHARED/
LIBRARY
(Title of Stack)
This library stack is shared by all users of BANKTR who have invoked the TRHISTORY journal. All transactions audited in the TRHISTORY journal are processed against the data base.
D4
TRHISTORY/
PRIVATE/
LIBRARY
(Title of Stack)
This library stack is private. One private stack is initiated for each invocation of the transaction library.
User-written Program
P1

While program P1 is running, a program named P2 is initiated. If P2 also processes transactions (which have been defined within transaction base BANKTR) against the data base, and thus against the TRHISTORY journal, then after P2's first call on a Transaction Library entry point the stack structure shown in the following diagram would result:

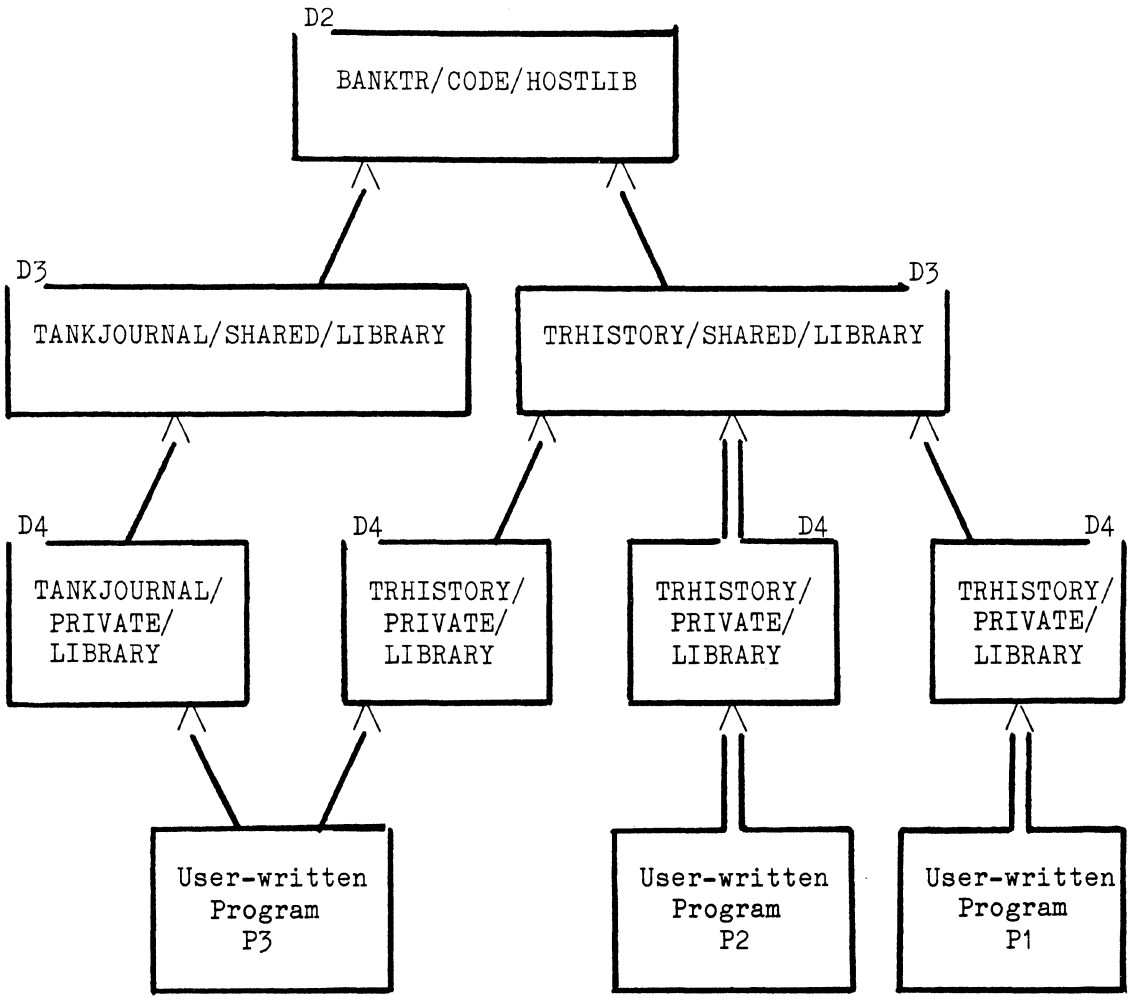
Transaction Library



If a third user-written program named P3 is initiated to tank transactions (which have been defined within transaction base BANKTR) to the transaction journal TANKJOURNAL, then the stack structure illustrated in the following diagram would result:



If program P3 then invokes the TRHISTORY transaction journal, the stack structure illustrated by the following diagram would result. As the diagram shows, program P3 invokes two different transaction journals.



The following examples show how to set the LIBPARAMETER library attribute in the user languages. In each of these examples, the name of the Transaction Library code file is BANKTR/CODE/HOSTLIB.

ALGOL:

```
LIBRARY L (TITLE = "BANKTR/CODE/HOSTLIB.");
          .
          .
          .
L.LIBPARAMETER:="TRHISTORY";
```

COBOL/COBOL74:

```
CHANGE ATTRIBUTE LIBPARAMETER OF
"BANKTR/CODE/HOSTLIB" TO "TRHISTORY."
```

DEFINITION OF TRANSACTION USER

A "transaction user" is the source of a set of transaction records for a particular journal. A transaction user can be a program or one of several terminals serviced by a program, or both. The transaction user concept is intended primarily to help in restarting user-written programs.

Each transaction user is identified by an alphanumeric string of characters that is unique among all transaction users of a particular journal. A transaction user must become known to a transaction journal before any processing can occur for that user. A transaction user becomes known to a journal when a user-written program calls the Transaction Library entry point CREATETRUSER. (The Transaction Library entry points mentioned in this discussion are described under "Entry Points" in this section.) The user-written program passes the transaction user's identifying string to CREATETRUSER. CREATEUSER, in turn, verifies that the transaction user is not already known to the transaction journal and assigns a unique three-digit integer value to represent the transaction user. This unique integer value can then be used by other Transaction Library entry points.

The character string that identifies a transaction user must be an EBCDIC string of at most 17 characters. The string should not include special characters, though this is not a requirement of the system.

A transaction user remains known to a particular journal until a user-written program calls the Transaction Library entry point PURGETRUSER. PURGETRUSER accepts the unique integer value that identifies a transaction user and eliminates all references to that user within that transaction journal. The Transaction Library rejects any later reference to a purged transaction user until that user is recreated.

In addition to creating a transaction user, the user-written program must activate the transaction user before any processing with that user can occur. Only one user-written program can activate a particular transaction user at any one time. This restriction ensures that no two user-written programs can simultaneously process transactions for the same transaction user. As a result, the response, input, and restart transaction records can be kept consistent for each transaction user.

To activate a transaction user, the user-written program must pass the identifying string of a known transaction user (hereafter referred to as the IDSTRING) to the Transaction Library entry point LOGONTRUSER. If

the transaction user is not already active, LOGONTRUSER marks the transaction user as active and returns to the user-written program the original integer value (hereafter referred to as the IDNUM) associated with that transaction user. Only active transaction users can submit transaction records to the Transaction Library. A transaction user becomes inactive when the user-written program terminates, or when the program calls one of these Transaction Library entry points: PURGETRUSER, LOGOFFTRUSER, or CLOSETRBASE.

A program can make several transaction users active by calling LOGONTRUSER several times, each time with a different IDSTRING. The program is then responsible for passing the appropriate IDNUM value whenever it calls one of the Transaction Library entry points requiring that value as a parameter.

The system saves the IDNUM parameter in the journal control file for each transaction user. In addition, during update processing a reference to the restart transaction record and the response transaction record corresponding to the last transaction is saved in the control file of the TRHISTORY journal for each transaction user. For tanking, a reference to the last tanked transaction record is saved in the control file of the tank journal for each transaction user. These records can be retrieved at any time to facilitate job restart.

NOTE

Transaction users are known to a journal but not to the entire Transaction Processing System. A particular IDSTRING may be known to one journal and not another. If a particular IDSTRING is known to more than one journal, the IDNUM value is unlikely to be the same for all journals. The maximum number of users for any journal is 128.

ENTRY POINTS

The following paragraphs describe the external entry points to the Transaction Library, which are called by user-written programs. These entry points provide the only means whereby the procedures of the Transaction Library can be invoked. For each entry point, a description of its function and its Transaction Library declaration are provided.

Each of these entry points returns a nonzero result (as the value of the procedure) if the Transaction Library detects an exception condition. The value returned by an entry point can be examined to determine the nature of the exception. (For further details about exception handling, the reader should refer to Section 8 of this manual.)

OPENTRBASE

OPENTRBASE initiates transaction processing and opens the transaction journal specified by the Transaction Library's LIBPARAMETER attribute. Opening the the transaction journal prepares it for subsequent use. OPENTRBASE must be the first Transaction Library entry point called by a user-written program. The entry point is declared in the Transaction Library as follows:

```
INTEGER PROCEDURE OPENTRBASE(USEROPTION, TIMEOUT);
    INTEGER USEROPTION, TIMEOUT;
```

OPENTRBASE requires two parameters. The TIMEOUT parameter is used only by the remote version of the Transaction Library. TIMEOUT indicates the time, in seconds, that the remote system waits for the host system to recover if the host system fails. If that time is exceeded, an exception is returned to the program, indicating that the host system is down.

The USEROPTION parameter determines which entry points the user-written program can call. Values for USEROPTION are interpreted in the following paragraphs. The integer values must be passed because the names are unknown to the compiler in this context.

Value	Meaning
1	Update Transaction records are to be processed, or applied to the data base, and all entry points except TANKTRANSACTION, TANKTRNORESTART, READTRANSACTION, and SEEKTRANSACTION can be called. This mode is permitted only for the TRHISTORY journal.
2	Inquiry Only data base inquiry can be performed; that is, only transactions that do not perform data base update operations can be used. When Inquiry is used, all entry points except TANKTRANSACTION, TANKTRNORESTART, READTRANSACTION, and SEEKTRANSACTION can be called. Update transactions that attempt to execute the user language statement BEGINTRANSACTION terminate abnormally. This mode is permitted only for the TRHISTORY journal.
3	Tank Transaction records are to be tanked (that is, written) to a tank journal and saved for future data base updates. When Tank is used, all entry points except PROCESSTRANSACTION, PROCESSTRNORESTART, PROCESSTRFROMTANK, READTRANSACTION, RETURNLASTRESPONSE, and SEEKTRANSACTION can be called. This mode is not permitted for the TRHISTORY journal.
4	Read Transaction records are to be read from the journal. All entry points except CREATETRUSER, PURGETRUSER, TANKTRANSACTION, TANKTRNORESTART, PROCESSTRNORESTART, PROCESSTRFROMTANK, PROCESSTRANSACTION, and SWITCHTRFILE can be called. While a journal is in this mode, any other process can open the journal. A program using read mode can simultaneously process against the same journal as programs using update, inquiry, tank, or exclusive update mode, or other programs using read mode. The journal is not actually opened until the entry point SEEKTRANSACTION is called to specify the file number and position. This mode is permitted for all transaction journals.
5	Exclusive Update Transaction records are to be processed or applied to the data

base. This parameter differs from the Update parameter in that this parameter allows only one updater. That is, only one program at a time can open the data base for Exclusive Update. When Exclusive Update is used, all entry points except TANKTRANSACTION, TANKTRNORESTART, READTRANSACTION, and SEEKTRANSACTION can be called. Exclusive Update is useful for large batch updating where transactions are not guaranteed to be independent of all others and, therefore, must run alone. This mode is permitted only for the TRHISTORY journal.

CREATETRUSER

CREATETRUSER creates and later identifies a new transaction user for the currently open journal. This entry point is declared in the Transaction Library as follows:

```
INTEGER PROCEDURE CREATETRUSER(IDSTRING, IDNUM);
    STRING IDSTRING;
    INTEGER IDNUM;
```

IDSTRING, a unique EBCDIC string of at most 17 characters, is input from the user-written program. This string must uniquely identify a user that is not already defined for this journal. A unique integer value (limited to (2**40)-1) that is used to represent IDSTRING is returned in the IDNUM parameter.

PURGETRUSER

PURGETRUSER purges or deletes a transaction user previously created by CREATETRUSER. This entry point is declared in the Transaction Library as follows:

```
INTEGER PROCEDURE PURGETRUSER(IDNUM);
    INTEGER IDNUM;
```

The user to be purged is identified by IDNUM, the input parameter. After PURGETRUSER is called, the transaction user is no longer known to the currently open journal, and information about this user's transactions is discarded.

TRUSERIDSTRING

TRUSERIDSTRING returns, in the parameter IDSTRING, the user identification string that corresponds to the value of the input parameter IDNUM. This entry point is declared in the Transaction Library as follows:

```
INTEGER PROCEDURE TRUSERIDSTRING(IDSTRING, IDNUM);
    INTEGER IDNUM;
    STRING IDSTRING;
```

LOGONTRUSER

LOGONTRUSER is called by a user-written program to make a transaction user active. The entry point is declared in the Transaction Library as follows:

```
INTEGER PROCEDURE LOGONTRUSER(IDSTRING, IDNUM);
    STRING IDSTRING; INTEGER IDNUM;
```

IDSTRING, an EBCDIC string of at most 17 characters, is input from the user-written program. This string must uniquely identify a transaction user that is already defined for this journal by a previous call on CREATETRUSER, but not yet active. The original integer value assigned to this IDSTRING is returned by the parameter IDNUM. Only one user of a given IDNUM can be active at a time.

LOGOFFTRUSER

LOGOFFTRUSER deactivates a transaction user. The entry point is declared in the Transaction Library as follows:

```
INTEGER PROCEDURE LOGOFFTRUSER(IDNUM);
    INTEGER IDNUM;
```

IDNUM must represent an active transaction user for this program.

RETURNLASTADDRESS

RETURNLASTADDRESS returns the address of the last transaction to be either tanked or processed by a transaction user. An address is composed of a data file number, the beginning segment address of a block within a data file, and a word offset that references the first word of the transaction record within the block. This entry point is declared in the Transaction Library as follows:

```
INTEGER PROCEDURE RETURNLASTADDRESS (FILENUM, BLOCKNUM, OFFSET,
    IDNUM);
    REAL FILENUM, BLOCKNUM, OFFSET;
    INTEGER IDNUM.
```

IDNUM must represent an active transaction user for this program. The address is returned in the three parameters FILENUM, BLOCKNUM, and OFFSET. The size of the FILENUM parameter is four digits, and the sizes of BLOCKNUM and OFFSET are determined by the way they are defined in the TRDESCRIPTION file.

RETURNLASTADDRESS allows transactions to be easily detanked. For example, RETURNLASTADDRESS could be called after tanking one transaction and called again after tanking is complete. At a later time, the entry point SEEKTRANSACTION could be called to position within a journal at the first address returned by RETURNLASTADDRESS,

after which the entry point READTRANSACTION could be called to read the previously tanked transaction records.

RETURNRESTARTINFO

RETURNRESTARTINFO is intended to help restart a user-written program. This entry point is declared in the Transaction Library as follows:

```
INTEGER PROCEDURE RETURNRESTARTINFO(IDNUM, TRREC);
    INTEGER IDNUM;
    TRANSACTION RECORD (TRBASE) TRREC;
```

IDNUM must represent an active transaction user for this program. The restart transaction record corresponding to the last transaction that was processed or tanked normally for user IDNUM is returned as the parameter TRREC. If no restart transaction record exists, TRREC contains the last input transaction that was processed or tanked normally for this user. If no input transaction record exists for this user, a value of 3 is returned.

RETURNLASTRESPONSE

RETURNLASTRESPONSE returns the last saved response transaction record for the user. This entry point is declared in the Transaction Library as follows:

```
INTEGER PROCEDURE RETURNLASTRESPONSE(IDNUM, TRREC);
    INTEGER IDNUM;
    TRANSACTION RECORD (TRBASE) TRREC;
```

The response transaction record corresponding to the last normally completed transaction for user IDNUM is returned as the parameter TRREC. If no response transaction record exists for the user, a result indicating this condition is returned.

Although RETURNLASTRESPONSE can aid in restarting a user-written program, the response record should not be used as the sole source of restart data because it may not be the last returned record. For reliable program restarting, the response record returned from RETURNLASTRESPONSE should be used in conjunction with the restart or input transaction record returned from the entry point RETURNRESTARTINFO.

PROCESSTRANSACTION

PROCESSTRANSACTION sends an input transaction record (TRIN) to the user's Update Library for processing against the data base. This entry point is declared in the Transaction Library as follows:

```
INTEGER PROCEDURE PROCESSTRANSACTION(IDNUM, TRIN, TROUT,
    RESTARTTR);
    INTEGER IDNUM;
    TRANSACTION RECORD (TRBASE) TRIN, TROUT, RESTARTTR;
```

IDNUM must represent an active transaction user for this program. A response transaction record is returned to the program as the parameter TROUT. If the transaction is successful, the exception flag returned by the entry point is 0, and the response transaction record is that created by the Update Library. Otherwise, the response is undefined and the exception value that is returned indicates the nature of the error. (Refer to section 8 of this manual for more information about exceptions.)

A successful transaction is one that appears to observe all requirements of TPS. In fact, however, such a transaction may not have been successful from the standpoint of the user-written program. In this situation, the response transaction should indicate that a problem has arisen.

The parameter RESTARTTR is a transaction record that contains user-defined restart information. RESTARTTR is saved for the user and can be retrieved by calling the entry point RETURNRESTARTINFO if a program restart is necessary. Both TRIN and RESTARTTR are audited in the TRHISTORY journal.

PROCESSTRNORESTART

PROCESSTRNORESTART performs the same function as PROCESSTRANSACTION, except that no restart transaction record is passed and subsequently audited in the TRHISTORY journal. This entry point is declared in the Transaction Library as follows:

```
INTEGER PROCEDURE PROCESSTRNORESTART(IDNUM, TRIN, TROUT);
    INTEGER IDNUM;
    TRANSACTION RECORD (TRBASE) TRIN, TROUT;
```

For PROCESSTRNORESTART, only the input transaction TRIN is saved in the TRHISTORY journal.

PROCESSTRNORESTART should be used to process transactions against the data base when the user-written program does not require the use of a restart transaction record.

PROCESSTRFROMTANK

PROCESSTRFROMTANK performs a function similar to PROCESSTRANSACTION, except that a transaction user number other than that of the input transaction is used to restart programs. PROCESSTRFROMTANK is used primarily for processing transactions from a tank file. This entry point is declared in the Transaction Library as follows:

```
INTEGER PROCEDURE PROCESSTRFROMTANK(IDNUM, TRIN, RESTARTNUM,
    RESTARTTR);
    INTEGER IDNUM, RESTARTNUM;
    TRANSACTION RECORD (TRBASE) TRIN, RESTARTTR;
```

IDNUM is assigned to TRIN, and RESTARTNUM is assigned to RESTARTTR. RESTARTNUM must be a known user of the TRHISTORY journal and must be active at the time of the entry point call. IDNUM need not be active when calling PROCESSTRFROMTANK. When transactions from a tank file are processed, the response transaction TROUT is not returned because it is of no interest to the detanking procedure.

TANKTRANSACTION

TANKTRANSACTION tanks an input transaction record and restart transaction record. This entry point is declared in the Transaction Library as follows:

```
INTEGER PROCEDURE TANKTRANSACTION(IDNUM, TRIN, RESTARTTR);
    INTEGER IDNUM;
    TRANSACTION RECORD (TRBASE) TRIN, RESTARTTR;
```

IDNUM must represent an active transaction user for the program that calls this entry point. RESTARTTR is a transaction record that is intended to contain user-defined restart information. The restart transaction record can be retrieved by calling the RETURNRESTARTINFO entry point if program restart is necessary. Both the input transaction TRIN and RESTARTTR are audited in the appropriate tank journal. The response transaction that is saved in the journal control file is that created by the Transaction Library. This response record cannot be accessed by user-written programs.

TANKTRNORESTART

TANKTRNORESTART performs the same function as TANKTRANSACTION except that no restart transaction record is passed and subsequently audited in the tank journal. For TANKTRNORESTART, only the input transaction TRIN is saved in the tank journal. TANKTRNORESTART should be used to tank transactions when the use of a restart record is not required. This entry point is declared in the Transaction Library as follows:

```
INTEGER PROCEDURE TANKTRNORESTART (IDNUM,TRIN);
    INTEGER IDNUM;
    TRANSACTION RECORD (TRBASE) TRIN;
```

READTRANSACTION

READTRANSACTION reads the next transaction record in sequence from the transaction journal and returns the record in the parameter

TRREC. This entry point is declared in the Transaction Library as follows:

```
INTEGER PROCEDURE READTRANSACTION(TRREC);
    TRANSACTION RECORD (TRBASE) TRREC;
```

If the transaction record has been marked as rejected, then an exception flag (in the ATTENTION error group) is returned along with the record. If no more transaction records remain, an end-of-file exception result is returned instead of a record.

READTRANSACTION can be called only after a call on the entry point SEEKTRANSACTION has opened and positioned the current record pointer within a specific journal data file. (For more information regarding exception processing, the reader should refer to Section 8 in this manual.)

SEEKTRANSACTION

SEEKTRANSACTION positions a current record pointer at a particular address within a journal data file. This entry point is declared in the Transaction Library as follows:

```
INTEGER PROCEDURE SEEKTRANSACTION(TRFILE, TRBLOCK, TROFFSET);
    INTEGER TRFILE, TRBLOCK, TROFFSET;
```

The parameters represent the address (file, block, and offset) of a transaction record in the transaction journal. The current record pointer is set so that the next call on READTRANSACTION reads the transaction record at that address. If TRBLOCK = 0, then the first record in the given file is selected. TROFFSET is ignored in this case. If TRBLOCK > 0 and TROFFSET = 0, then the first record in the given block is selected. TRFILE must always specify a valid file number.

Except for the cases described in the preceding paragraph, the parameters must refer exactly to the left control word of a valid transaction record; otherwise, no seek is performed and an exception is returned. SEEKTRANSACTION must be called before the first READTRANSACTION call in order to open a file and position the current record pointer within that file.

The address of a particular transaction can be obtained by using the TRUTILITY SEARCH function. TRUTILITY prints the addresses of any transactions that are located according to the search specification. For more details, the reader should refer to SEARCH Statement in Section 11 of this manual.

SWITCHTRFILE

SWITCHTRFILE forces a file switch on the current data file of the journal. The current file is closed, the file number associated with the current file is incremented by 1, and the next file in sequence is created. The next write to the journal occurs on the new file.

If SWITCHTRFILE is not called, the Transaction Library creates the next journal data file in sequence when the current file becomes full. This entry point is declared in the Transaction Library as follows:

```
INTEGER PROCEDURE SWITCHTRFILE;
```

HANDLESTATISTICS

HANDLESTATISTICS allows users to print out all TPS statistics and to reset the statistics while the transaction base is open. (refer to TPS STATISTICS in this section for more information). This entry point is declared in the Transaction Library and the Remote Library as follows:

```
INTEGER PROCEDURE HANDLESTATISTICS(STATOPTION);
    VALUE STATOPTION;
    INTEGER STATOPTION;
```

Values for the STATOPTION parameter are interpreted as follows:

Value	Meaning
1	TPS statistics are printed out and then reset.
2	TPS statistics are printed out.
3	TPS statistics are reset.

CLOSETRBASE

CLOSETRBASE ends the use of TPS by the calling program. All active transaction users for the program are marked inactive, and all subsequent calls on Transaction Library entry points are illegal until OPENTRBASE is called again. CLOSETRBASE is declared in the Transaction Library as follows:

```
INTEGER PROCEDURE CLOSETRBASE;
```

TRANSACTION LIBRARY DIAGNOSTIC MESSAGES

If an error condition occurs while a transaction is being processed, the Transaction Library displays diagnostic messages to indicate any of the following conditions:

1. The Transaction Library is forcing the user program to exit the Update Library.

2. The Transaction Library, rather than the Update Library, is forcing an ABORT to occur.

3. The Transaction Library is stopping other users of the journal in order to allow transactions to be reprocessed.

If message display at the terminal is not desired, the user should use the CANDE RO command to reset the station option MESSAGE. (Refer to the CANDE Reference Manual for the use of this command.)

TPS STATISTICS

The Transaction Library can collect and produce a summary of TPS performance statistics for each transaction journal that it opens. These statistics are produced only if the TFL <parameters> option STATISTICS is set for the appropriate transaction base (refer to "Parameters" in Section 3 of this manual), and if the transaction journal is opened for update, inquiry, tanking, or exclusive update (refer to OPENTRBASE under "Entry Points" in this section).

The amount and type of information included within the summary varies depending on the type of processing undertaken. For example, no statistics regarding the performance of the Update Library are included for a tank journal. The user cannot get a printout of performance statistics until the <transaction journal name>/SHARED/LIBRARY (D3) stack terminates.

The following information may be included in a statistical summary, depending on the type of processing. Each of the time-related statistics is measured in terms of elapsed time.

Base Name and Journal Name

The name of the transaction base and transaction journal for which the statistics are collected.

Time of Collection

The date and time that collection of statistics began and ended, and the amount of time that elapsed during the collection interval.

Journal Addresses

The journal's data file number and data block number (beginning segment address), and the data block serial number at which processing or tanking of transactions began and ended.

TRHISTORY Journal Statistics

If the TRHISTORY journal is being used, then the following statistics are output:

1. Total user transactions.

   This statistic gives the total number of user transactions submitted to the Transaction Library. A user transaction is defined as a single input transaction record. Restart transaction records do not affect this total.

2. Update transactions.

   This statistic gives the total number of user update transactions. An update transaction is defined as a transaction that enters transaction state when applied to the data base.

3. Inquiry transactions.

   This statistic gives the total number of user inquiry transactions. An inquiry transaction is defined as a transaction that does not enter transaction state when applied to the data base.

4. Total system transactions.

   This statistic gives the total number of system transactions generated by the Transaction Library. System transactions are audited to transaction journal data files at various points during processing and are used only by TPS software.

Tank Journal Statistics

If a tank journal is being used, the following statistics are output:

1. Total user transactions.

   This statistic gives the total number of user transactions submitted to the Transaction Library. A user transaction is defined as a single input transaction record. Restart transaction records do not affect this total.

2. Number of tanked transactions.

   This statistic gives the total number of user transactions that were tanked. This number is equal to the total number of user transactions submitted to the Transaction Library.

3. Total system transactions.

   This statistic gives the total number of system transactions generated by the Transaction Library. System transactions are audited to transaction journal data files at various points during processing, and are used solely by TPS software.

Total Time Per User Transaction

This statistic gives the average total time, in milliseconds, per user transaction. This time is the average total time needed to process or tank a user transaction, measured from the time a transaction enters the Transaction Library to the time it exits from the Transaction Library.

Writes to Journal Data File

This statistic includes the following items:

1. Number of physical writes.

   This statistic gives the total number of physical writes performed against the journal's data file(s). If this number is significantly larger than the total number of user transactions, it indicates that the actual data block size is too small.

   The actual data block size is the maximum size allowed for a data block that is written to the journal's data file(s). This value is derived from the specified or default value for the journal's BLOCKSIZE attribute. The value of BLOCKSIZE is either given explicitly in the <transaction journal> declaration in the TFL source or obtained from the TFL <defaults> declaration or system default.

2. Number of non-writes.

This statistic gives the total number of non-writes detected for the journal. The number of non-writes associated with a single data block written to the journal is defined as the number of transactions completely contained within that block, minus 1. The total number of non-writes is computed by summing the number of non-writes for all of the data blocks written to the journal. In this context, a transaction is considered to be either an input transaction record, or an input transaction record and its corresponding restart transaction record (created as one transaction).

The number of non-writes provides a measure of the transaction record blocking factor. As the number of non-writes increases, the number of data blocks containing more than one transaction increases; as a result, the number of physical writes to the journal data file(s) decreases. The decrease in the number of physical writes should result in an increase in throughput.

As a general guideline, the number of non-writes should increase in proportion to the number of transactions that are submitted to the Transaction Library.

3. Average block size written.

This statistic gives the average size, in words, of all data blocks written to the journal's data file(s). If the number of non-writes is small and the average data block size written is the same as, or close to, the actual data block size (defined previously in "a"), it may be an indication that the block size of the journal data file is too small. The average data block size is output only if data has been written to the journal's data file(s).

4. Actual block size.

This statistic gives the maximum size, in words, that is allowed for data blocks written to the journal's data file(s). This value is derived from the file attributes specified for the journal in the TFL source. The actual data block size is only output if data is written to the journal's data file(s).

5. Average I/O wait.

This statistic gives the average time, in milliseconds, spent waiting for a physical write to complete against the primary journal data file. This statistic is output only if at least one physical write has occurred.

6. Average I/O wait (duplicated data file).

This statistic gives the average time, in milliseconds, spent waiting for a physical write to complete against the secondary journal data file. This statistic is output only if at least

one physical write has occurred and the journal contains duplicated data files.

Writes To Journal Control File

This statistic includes the following items:

1. Number of responses saved.

   This statistic gives the total number of response transaction records that are saved in the journal's control file. A response transaction record is saved for every update transaction and for any inquiry transactions that call procedure SAVERESPONSE. (SAVERESPONSE is described under PROGRAMMING THE UPDATE LIBRARY in Section 6 of this manual.) This statistic is output only if at least one response transaction record has been saved.

2. Average response save time.

   This statistic gives the average time, in milliseconds, required to save a response transaction record in the journal's control file. The average time used to save a response transaction is defined as the time required to initiate the response transaction record write(s), plus the time required to wait for the write(s) to complete. This statistic is output only if at least one response transaction record has been saved.

3. Average block size written.

   This statistic gives the average size, in words, of all data blocks written to the journal's control file. This statistic is output only if at least one response transaction record has been saved.

4. Actual block size.

   This statistic represents the maximum size, in words, allowed for data blocks that are written to the journal's control file. This value is derived from the file attributes specified for the journal in the TFL source. The actual block is output only if at least one response transaction record has been written to the journal's control file.

Calls on Update Library

This statistic gives the total number of times that the Transaction Library has called the user's Update Library in order to process a transaction. This statistic is output only if at least one transaction has been applied to the data base.

Average Time in Update Library

This statistic gives the average total time, in milliseconds, that was used for processing within the user's Update Library. This statistic is defined as the average time (for all transactions applied to the data base) used for processing within the Update Library. Processing time within the Update Library is measured from the time the Transaction Library calls the Update Library to the time the Transaction Library returns from the call, minus the time spent executing the Transaction Library procedures SAVEINPUT and SAVERESPONSE. (For more information about these procedures, the reader should refer to Section 6 of this manual.) This statistic is output only if at least one transaction has been applied to the data base.

Number of ABORTs

This statistic gives the number of data base ABORTs forced by the Transaction Library. This statistic is output only if at least one data base ABORT has been forced by the Transaction Library.

Average ABORT Recovery Time

This statistic gives the average time, in seconds, that is required for a data base ABORT to complete. This statistic is output only if at least one data base ABORT has been forced by the Transaction Library.

Total Transactions Reprocessed

This statistic gives the total number of transactions that were reprocessed by the Transaction Library.

6 UPDATE LIBRARY

The Update Library is a collection of user-written transaction processing routines that serve as an interface between the Transaction Library and a DMSII data base. These processing routines can be written in any one of the DMSII user languages: ALGOL, COBOL, or COBOL74. The Update Library is the only user-written module within TPS that contains the data base declaration and all of the code that performs data management statements against the data base.

To ensure effective interaction between the Update Library and the Transaction Library, the user must follow certain conventions in regard to data base consistency and transaction reproducibility when programming the Update Library. DMSII provides a tool known as "record level locking" for use with the transaction routines so that consistency can be preserved in a multiprogramming environment. For record level locking to preserve consistency, the user must observe a programmatic convention known as the "two-phase transaction," which in turn makes transaction reproducibility easier.

Consistency, record level locking, the two-phase transaction, and reproducibility are thoroughly explained in later parts of this section entitled "Data Base Consistency" and "Reproducibility." This section also discusses three possible methods of structuring and programming the Update Library.

PROGRAMMING THE UPDATE LIBRARY

Although the Update Library is user-written, it must conform to certain conventions established by TPS so that it can be accessed by the Transaction Library. These conventions are described in the following paragraphs.

The Update Library is required to provide one entry point that makes it accessible to the Transaction Library. The name of this entry point is ACCESSDATABASE. For COBOL and COBOL74 Update Libraries, ACCESSDATABASE must be the name of the PROGRAM-ID in the IDENTIFICATION DIVISION. For ALGOL Update Libraries, ACCESSDATABASE must be the name of the procedure that is the entry point.

The ACCESSDATABASE entry point accepts the following parameters, which are listed in the order in which they must be declared:

1. A function flag that indicates which basic function the Update Library should perform. This value is input to the Update Library from the Transaction Library. The Update Library is

not restricted as to how these functions are implemented, but the following conventions for the values of the function flag must be observed. No other function flag values are valid.

The names given to the function flag values are used to identify the functions as they are discussed in the remainder of this section. These functions can be identified by other names or simply by their integer values.

Integer Value	Name and Function
1	OPENDATABASE for update This value causes the Update Library to open either the data base or the appropriate portion of the data base for updating.
2	OPENDATABASE for inquiry This value causes the Update Library to open either the data base or the appropriate portion of the data base for inquiry.
3	UPDATE This value causes the Update Library to process the transaction update routine associated with the input transaction record. This input transaction record is passed as the second parameter to ACCESSDATABASE.
4	FORCEABORT This value causes the Update Library to close either the data base or a portion of the data base in order to cause a data base ABORT.
5	CLOSEDATABASE This value causes the Update Library to close all portions of the data base. This function is requested if the user-written program called the CLOSETRBASE Transaction Library entry point, or if the Transaction Library requested the close because the user program did not call CLOSETRBASE.

2. An input transaction record that contains input data for one of the transaction update routines. The appropriate input routine is chosen based on the format, and possibly the subformat, that

is contained within the transaction record. This record is referred to as "TRIN" in the following paragraphs.

3. An output transaction record that contains the data output from a transaction update routine. This record is also known as the "response transaction record" and is referred to as "TROUT" in the following paragraphs.

4. A Transaction Library procedure named SAVEINPUTTR that is passed as a formal parameter to the Update Library. This procedure is used in the DMSII user language MIDTRANSACTION statement to save transaction records in the TRHISTORY journal data files. For more information about MIDTRANSACTION, refer to "DMSII User Language Syntax For Transaction Processing" in this section.

5. A Transaction Library procedure named SAVERESPONSETR that is passed as a formal parameter to the Update Library. This procedure is used in the TPS version of the DMSII user language ENDTRANSACTION statement to save the output transaction (TROUT) in the TRHISTORY journal control file. For more information about the ENDTRANSACTION statement, refer to "DMSII User Language Syntax For Transaction Processing" in this section.

Parameter Passing to a COBOL74 Update Library

Update Libraries written in COBOL and COBOL74 can be exactly the same, except for the declaration of INTEGER parameters. For an Update Library written in COBOL74 to receive INTEGER parameters from Transaction Library entry points (written in ALGOL), the parameters must be declared as follows:

Examples:

Method 1.

Declaration in Linkage section:

```
77  FUNCTIONFLAG COMP PIC 9(8)...
```

Use of Parameter in Procedure Division Header:

```
PROCEDURE DIVISION USING INTEGER(FUNCTIONFLAG),
    .....
```

Method 2.

Declaration in Linkage section:

```
77  BINARYFUNCTIONFLAG BINARY PIC 9(8).
```

Use of Parameter in Procedure Division Header:

```
PROCEDURE DIVISION USING BINARYFUNCTIONFLAG,
    .....
```

Methods of Structuring the Update Library

This section discusses the following approaches to structuring the Update Library, and provides guidelines for using each approach:

1. Invoking the entire data base using a single update library.
2. Invoking part of the data base using a single update library.
3. Invoking multiple parts of the data base using multiple update libraries.

Invoking The Entire Data Base Using A Single Library

If this method of structuring the Update Library is used, the functions that the ACCESSDATABASE entry point provides must be used as described in the following paragraphs. For this method, the Update Library is a PRIVATE library.

OPENDATABASE (Update or Inquiry)

Each function must open the entire data base. In this case, the OPEN option for update must be TRUPDATE. When opening the entire data base, the option for update in the user's OPEN statement must be TRUPDATE. Refer to "DMSII User Language Syntax For Transaction Processing" in this section for further details.

UPDATE (TRIN, TROUT, SAVEINPUTTR, SAVERESPONSETR)

This function is called by the Transaction Library to process an input transaction record (TRIN) that has been passed from the user-written program as a parameter to the entry point PROCESSTRANSACTION, PROCESSTRNORESTART, or PROCESSTRFROMTANK. The function must generate a response transaction record (TROUT), which is passed back through the Transaction Library to the user program.

A user routine that processes the input transaction record (TRIN) must execute, at most, a single BEGINTRANSACTION/ENDTRANSACTION pair. All Update Library routines must use the same locking strategy: either two-phase locking or an equivalent programmatic convention. The records should remain locked until they are freed by ENDTRANSACTION. (Refer to "Data Base Consistency" and "Reproducibility" in this section for further details.)

The UPDATE procedure should be structured so that the input transaction is processed based on its format and subformat type. The following is a sample UPDATE procedure written in ALGOL:

```
PROCEDURE UPDATE(TRIN, TROUT, SAVEINPUT, SAVERESPONSE);
...
BEGIN
     ...
     CASE TRIN.TRFORMAT OF
     BEGIN
(<format 1>):
               .
               .
(<format 2>):
               .
               .
(<format n>):
          CASE TRIN.TRSUBFORMAT OF
          BEGIN
     (<subformat 1>):
                    .
                    .
     (<subformat m>):
                    <process transaction with format n, subformat m>;
                    .
                    .
          END SUBFORMATS;
               .
               .
               .
     END FORMATS;
     ....


END;
```

FORCEABORT

When the entire data base is invoked, this procedure must close the entire data base. This procedure is called only when the data base is in transaction state and the Transaction Library detects that an ABORT is required. Under these conditions, the close operation causes a data base ABORT.

CLOSEDATABASE

This function is called to close the entire data base.

For this method of structuring the Update Library, the FORCEABORT and CLOSEDATABASE functions perform the same function.

Invoking Part Of The Data Base Using A Single Library

A second method of programming the Update Library is based on breaking up the data base into logical databases so that only a portion of the data base can be invoked. This approach can be useful when the data base is large. For this method, the Update Library is a PRIVATE library.

When invoking only part of the data base, the user need not conform to the restrictions that apply when opening the entire database, though the necessary checks must still be provided.

OPENDATABASE (Inquiry or Update)

The RESTART DATA SET and other desired portions of the data base must be opened here so that RECOVERY can run, if needed. If other procedures are used to open the logical data base, then only the RESTART DATA SET must be invoked here. The function flag indicating whether the data base was opened for inquiry or update should be saved so that subsequent opens (that is, calls to the procedures that open the data base) are performed consistently.

UPDATE (TRIN, TROUT, SAVEINPUTTR, SAVERESPONSETR)

Before processing an input transaction, the relevant portion of the data base must be invoked; otherwise, an INVALID OP error results. Because the entire data base is not opened using OPENDATABASE, the user must ensure that the needed portion is opened at some point before the input transaction is processed.

A suggested approach is to group transaction record formats into subbases of the transaction base that correspond to logical databases of the data base. Then transaction records in a given subbase refer only to the structures invoked in the corresponding logical data base.

The Update Library can use the subbase number in the control portion of the input transaction record being processed to call the appropriate procedure to handle that subbase. A global variable for each subbase can be kept to indicate whether the procedure is being called for the first time and, if so, whether the logical data base must be opened. In this way, only those portions of the data base that this user-written program uses are invoked, and they can be invoked as needed.

FORCEABORT

When the Transaction Library detects that the last transaction was not successfully completed in the Update Library, it calls FORCEABORT to close the portion of the data base that was used by the transaction. If multiple logical databases are open, the Update Library must have a global value to keep track of the logical data base to which the last transaction applied. Then if FORCEABORT is called, the Update Library knows which portion of the data base to close.

CLOSEDATABASE

The Update Library must be able to close all logical databases that have been invoked by the user-written program. Again, a global value for each logical data base is sufficient to determine which logical databases must be closed.

Invoking Parts Of The Data Base Using Multiple Libraries

A variation on invoking part of the data base is to use libraries to divide up the code associated with each logical data base. Because the Update Library is used to keep all of the code that accesses the data base logically in one code file, a large code file might be generated for the Update Library. In this situation, it might be necessary to break up the code file into pieces and use only the portions that are required at any given time.

The code file can be broken up easily by using libraries. The Update Library can contain library declarations for many other libraries. These libraries, in turn, can provide appropriate UPDATE, OPENDATABASE, FORCEABORT, and CLOSEDATABASE functions for groups of transaction records. The Update Library would then route each transaction to the library that is handling that particular transaction record format and, optionally, subformat. To simplify this process, related transaction record formats can be grouped together into subbases in the TFL source, and then each input transaction record can be routed to the appropriate library based on its subbase number.

If the user has related a transaction subbase to a logical data base, then the library that handles a particular subbase is only required to open one logical data base rather than opening the entire data base.

Because the subbase number is not implemented as a compile-time construct in the way that format numbers are, the actual number given to a particular subbase must be written into the program. The number 1 is always given to the entire transaction base. Thus, a transaction created when the entire transaction base is invoked is always assigned the number 1. Any changes in subbase numbers that result from adding or deleting subbases in a TFL update should be carefully noted.

Because the UPDATE procedure in the subbase libraries determines whether a subbase has been opened, the Update Library does not have to keep track of which subbases have been opened. Thus, the individual libraries need to provide only the UPDATE and FORCEABORT functions for the Update Library to use. In this case, the Transaction Library continues to call the ACCESSDATABASE procedure in the principal Update Library to perform the appropriate function. The principal Update Library is a PRIVATE library.

Example of Update Library

Whatever approach the user takes in implementing the Update Library, the library must provide the external entry point ACCESSDATABASE and must be compiled as <base name>/CODE/UPDATELIB so that the Transaction Library can find it.

An example of the correct structure for an Update Library is shown below in an ALGOL "skeleton" program. The example uses multiple libraries to provide the code that actually processes the transaction records. The code provided through these other libraries can be written in any language. (In addition to the example that follows, complete Update Libraries written in ALGOL and COBOL are presented in Appendix C of this manual.)

```
  $ SHARING = PRIVATE
 BEGIN  % TRANSACTION UPDATE LIBRARY

      LIBRARY DBSUBONE ( TITLE = "TRBASE/UPDATELIB/SUBONE ");

           PROCEDURE ACCESSSUBBASEONE(FUNCTIONFLAG,INQ,TRIN,TROUT,
                   SAVEINPUT, SAVERESPONSE );
                VALUE FUNCTIONFLAG, INQ;
                REAL FUNCTIONFLAG, INQ;
                TRANSACTION RECORD (TRB) TRIN, TROUT;
                PROCEDURE SAVEINPUTTR(); FORMAL;
                PROCEDURE SAVERESPONSETR(); FORMAL;
                LIBRARY DBSUBONE;

      LIBRARY DBSUBTWO ( TITLE = "TRBASE/UPDATELIB/SUBTWO ");

           PROCEDURE ACCESSSUBBASETWO(FUNCTIONFLAG,INQ,TRIN,TROUT,
                   SAVEINPUT,SAVERESPONSETR );
                VALUE FUNCTIONFLAG, INQ;
                REAL FUNCTIONFLAG, INQ;
                TRANSACTION RECORD (TRB) TRIN, TROUT;
                PROCEDURE SAVEINPUTTR(); FORMAL;
                PROCEDURE SAVERESPONSETR(); FORMAL;
                LIBRARY DBSUBONE;

      DEFINE UPDATEV = 1 #,
             FORCEABORTV = 2 #;

      % global variables
      REAL LASTSUBBASE, OPENTYPE;
      ....

      PROCEDURE FORCEABORT;
      BEGIN
```

```
        CASE LASTSUBBASE OF
        BEGIN
        ( 2 ) :
            ACCESSSUBBASEONE(FORCEABORTV,SAVEFUNCTIONFLAG,TRIN,TROUT,
                        SAVEINPUT,SAVERESPONSE);
      ( 3 ) :
            ACCESSSUBBASETWO(FORCEABORTV,SAVEFUNCTIONFLAG,TRIN,TROUT,
                        SAVEINPUT,SAVERESPONSE);
            ....
      END OF CASE;
      ...
END;

PROCEDURE UPDATE(TRIN,TROUT,SAVEINPUTTR,SAVERESPONSETR);
TRANSACTION RECORD (TRB) TRIN, TROUT;
PROCEDURE SAVEINPUTTR(); FORMAL;
PROCEDURE SAVERESPONSETR(); FORMAL;
BEGIN
      ....
      LASTSUBBASE := TRIN.TRSUBBASE;
      CASE TRIN.TRSUBBASE OF
      BEGIN
      ( 2 ) :
            ACCESSSUBBASEONE(UPDATEV,SAVEFUNCTIONFLAG,TRIN,TROUT,
                  SAVEINPUT,SAVERESPONSE);
            %  invokes library DBSUBONE
      ( 3 ) :
            ACCESSSUBBASETWO(UPDATEV,SAVEFUNCTIONFLAG,TRIN,TROUT,
                  SAVEINPUT,SAVERESPONSE);
            %  invokes library DBSUBTWO
            ...
      END OF CASES;
      ...
END;

PROCEDURE ACCESSDATABASE(FUNCTIONFLAG,TRIN,TROUT,SAVEINPUT,
            SAVERESPONSE);
      VALUE FUNCTIONFLAG;
      REAL FUNCTIONFLAG;
      TRANSACTION RECORD (TRB) TRIN, TROUT;
      PROCEDURE SAVEINPUT(); FORMAL;
      PROCEDURE SAVERESPONSE(); FORMAL;
% EXTERNAL ENTRYPOINT
BEGIN

      CASE FUNCTIONFLAG OF
      BEGIN
            1:    % OPEN UPDATE
                  OPENTYPE := FUNCTIONFLAG;
            2:    % OPEN INQUIRY
                  OPENTYPE := FUNCTIONFLAG;
            3:    % UPDATE
                  UPDATE(TRIN, TROUT, SAVEINPUT, SAVERESPONSE);
```

```
            4:    % FORCEABORT
                  FORCEABORT;
            5:    % CLOSE DATABASE
                  % LET BLOCKEXIT DO IT;
        END;
    END ACCESSDATABASE;

*************************************************************************


    EXPORT
        ACCESSDATABASE;

    FREEZE(TEMPORARY);


END OF LIBRARY.
```

DMSII USER LANGUAGE SYNTAX FOR TRANSACTION PROCESSING

Synchronization of TPS recovery with DMSII Recovery is an important aspect of TPS. To accomplish this, the following extensions to the present user language interface to DMSII have been defined:

1. The MIDTRANSACTION statement.

2. Optional extensions to the BEGINTRANSACTION and ENDTRANSACTION statements.

3. The TRUPDATE option for the OPEN statement.

DMSII applications that do not use TPS can continue to use the DMSII syntax that existed prior to the implementation of TPS. However, user-written code in the Update Library must use the syntax as it is defined here.

BEGINTRANSACTION, MIDTRANSACTION, and ENDTRANSACTION Statements

BEGINTRANSACTION, MIDTRANSACTION, and ENDTRANSACTION must all be able to pass a transaction record, as well as present parameters, to procedures in the ACCESSROUTINES. For the new forms only, NO-AUDIT is the default value for both BEGINTRANSACTION and ENDTRANSACTION; neither AUDIT nor NO-AUDIT can be specified with these forms. Refer to the DMSII User Language Interface Manual for more information concerning the BEGINTRANSACTION, MIDTRANSACTION, and ENDTRANSACTION statements.

SAVEINPUTTR and SAVERESPONSETR are untyped formal procedures with no parameters. The compiler generates a call on a particular procedure immediately prior to the call on the DMSII procedure in the ACCESSROUTINES.

COBOL/COBOL74 Syntax:

BEGIN-TRANSACTION (transaction-record-variable-id) restart-data-set-name

[exception-handling]

MID-TRANSACTION (mid-transaction-parameters) restart-data-set-name

[exception-handling]

END-TRANSACTION (end-transaction-parameters) restart-data-set-name

[SYNC] [exception-handling]

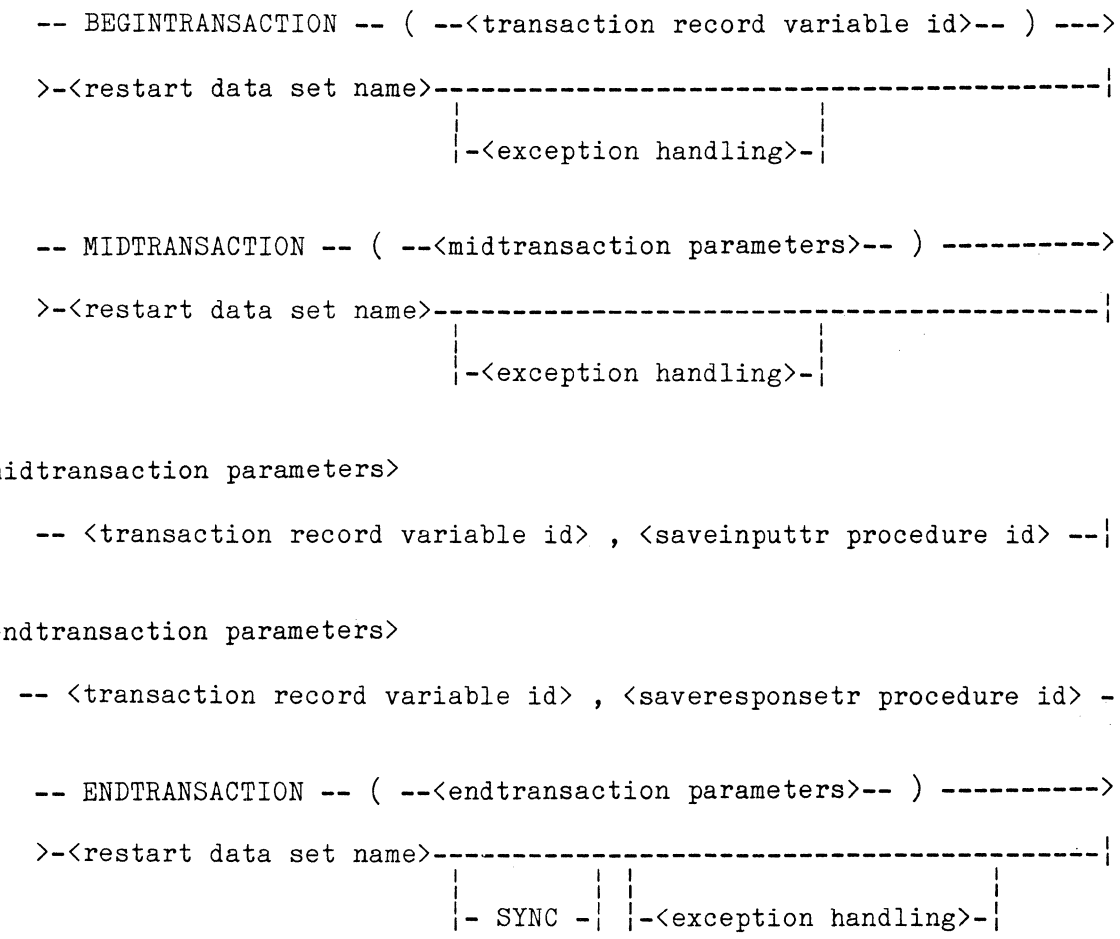
mid-transaction-parameters

transaction-record-variable-id, SAVEINPUTTR-procedure-identifier

end-transaction-parameter

transaction-record-variable-id, SAVERESPONSETR - procedure - identifier

ALGOL Syntax:



Semantics (for both languages):

<transaction record variable id>

The name of the formal input transaction record variable. (Refer to "Declaration of Transaction Record Variables" in Section 4 for a more detailed discussion of transaction record variables.)

<saveinputtr procedure identifier>

The name of the SAVEINPUTTR formal procedure.

<saveresponsetr procedure identifier>

The name of the SAVERESPONSETR formal procedure.

<exception>

Defined for the various languages in the DMSII User Language Interface Manual.

OPEN Statement # TRUPDATE Option

The TRUPDATE option for the OPEN statement must be used instead of the UPDATE option in the Update Library. The statements described in the preceding paragraphs are permitted only when the TRUPDATE option is used. Using the INQUIRY statement is permissible, but when INQUIRY is used, no forms of the BEGINTRANSACTION, MIDTRANSACTION, and ENDTRANSACTION statements are allowed.

Data Base Consistency And Update Library Programming

Every data base includes a set of assertions, or consistency constraints, that data within the data base must satisfy. Although few of these assertions are stated formally, they are extremely important. When the data preserves all the constraints, the data base is said to be "consistent"; otherwise, the data base is said to be "inconsistent."

For example, consider a DMSII data base that contains two data sets: a manager's data set containing the names of all the managers in a particular company along with their employee numbers, and an employee data set containing only employee numbers. Suppose the consistency constraint to be preserved is that an employee record exists for each manager's record. If the data base is consistent, then for every manager's record within the manager's data set, a corresponding employee record within the employee data set also exists. If a manager's record exists that does not have a corresponding record in the employee data set, then the data base is considered inconsistent.

Every transaction that updates a data base must observe the consistency constraints when run alone; otherwise, the transaction is considered incorrect. However, even if all transaction routines observe all the consistency constraints when run alone, this is no guarantee that all multiprogrammed mixes of these transactions also observe the consistency constraints. To ensure that consistency can be achieved in a multiprogramming environment, DMSII provides a tool known as "record level locking" for use by the Update Library's transaction routines.

In order for record level locking to preserve consistency, a programmatic convention known as the "two-phase transaction" must be observed. When an arbitrary mix of transaction routines is run concurrently, all transactions must be two-phase in order to preserve consistency. If all transactions are not two-phase, the transaction routines may execute in such a manner that consistency is not preserved. In fact, if all transactions are two-phase, no transaction ever violates the consistency constraints (except during transaction state, when processing temporarily invalidates one or more of the constraints).

A transaction is two-phase if it can be divided into a "growing" phase followed by a "shrinking" phase. During the growing phase, the transaction locks records but does not free any. During the shrinking phase, the transaction frees records but does not lock any. Additionally, if a record is updated, it must first be retrieved with an exclusive lock. Records that are retrieved but not updated can either be shared among readers or locked exclusively, but must be locked against retrieval by potential updaters.

Update Library

To ensure consistency of the data base, each transaction update routine within the Update Library must provide a locking strategy that causes the transaction to be two-phase or equivalent to two-phase. During the first part of an update routine, the appropriate records are locked and no records are freed. This part of the routine corresponds to the first phase of a two-phase transaction. Any access to the protected records after the last record is locked and until the first record is freed is considered to be in the midtransaction phase. The protected records are accessed only during the midtransaction phase. The midtransaction phase corresponds to the point between the phases of a two-phase transaction. The freeing of the appropriate records corresponds to the second phase of a two-phase transaction; no records can be locked during this phase.

Various alternatives for updating or examining many records are discussed in the following paragraphs. But before examining these alternatives, the ramifications of two-phase transactions for DMSII can be summarized as follows:

1. The user language FIND statement cannot be used at all. Only the LOCK statement can be used.

2. Because a LOCK statement obtains a record exclusively, records cannot be shared among readers.

3. Care must be taken, because DMSII executes an implicit FREE statement when appropriate before retrieving the requested record. Thus, a LOCK statement may be equivalent to a FREE statement followed by a LOCK statement, which is a violation of the two-phase condition. If more than one record in a data set must be locked, the data set may need to be invoked more than once because of the implicit FREE.

   A DELETE statement followed by a selection expression is equivalent to a LOCK statement followed by a DELETE <data set name> statement. Thus, care must also be taken in using DELETE, so that unwanted implicit FREE statements are not executed.

4. Because an ENDTRANSACTION statement frees all records, it makes execution of the next two-phase transaction easier.

5. A potential bottleneck may be caused if a record or set of records is locked by many transactions. (An example is a record containing information that is global to many transactions, like a bank total for a banking application.) The bottleneck can be reduced significantly if such a record is the last record locked in the transaction and it is locked in transaction state. The record can then be updated, stored, and immediately freed. This scheme keeps the record locked for the shortest possible time.

The Transaction Library does not enforce two-phase locking. If two-phase locking or an equivalent programmatic convention is not used, reproducibility of previous results and continued consistency of the data base probably cannot be achieved when RECOVERY operations complete.

The user routines should be programmed so that they do not depend on variables or DMSII paths from previous transactions. The routines should be programmed so that the input transaction record has all the information needed to process the transaction.

During normal processing the Update Library's internal variables are maintained between successive calls because it is only attached to one user program. This is true unless the multiple update libraries structuring method is chosen and one or more of the sub-libraries are SHAREDBYALL. In this situation, internal variables might not be maintained. However, after a DMSII Recovery occurs, the Transaction Library uses a single copy of the Update Library to reprocess backed out transactions. Therefore, this copy of the Update Library sees the requests for the various users interleaved when the Transaction Library reprocesses the backed out transactions.

For example, suppose the user writes a pair of Update Library routines. The first routine processes the first member of a data set, and the second is called repetitively to process successive members of the data set until no more remain. Finally, suppose that the data set is processed in the order of an index-sequential set, S, which has key item K and NO DUPLICATES. Then the second routine should not use FIND NEXT S because the path of the set S can be changed during transaction reprocessing by transactions interleaved from other users. Instead, the routine should use FIND S AT K > OLDVALUE, where OLDVALUE is an item in the input or restart transaction record that the calling user program sets equal to the value processed on the previous transaction. (The Update Library would have returned this value in the transaction output record.)

This technique makes the Update Library routine independent of the current value of the path for S. If "FIND NEXT S" were used, the routine would appear to work properly during normal processing. However, using this statement would probably prevent reproducibility after a RECOVERY.

Update Library routines must execute the BEGINTRANSACTION, MIDTRANSACTION, and ENDTRANSACTION statements in order; otherwise, TPS returns an error. The Update Library programmer is responsible for ensuring that the MIDTRANSACTION statement is, in fact, executed during the midtransaction phase of the transaction. The SAVEINPUT procedure must be specified in the MIDTRANSACTION statement and is implicitly called at that point. Similarly, the SAVERESPONSETR procedure must be

specified in the ENDTRANSACTION statement and is implicitly called at that point.

For inquiry transactions, the only kind permitted for inquiry mode, SAVERESPONSETR can be called explicitly. BEGINTRANSACTION, MIDTRANSACTION, and ENDTRANSACTION cannot be called. Thus, an Update Library routine for an update transaction should have a structure of the following form:

```
   .
   .
<lock all required records>
   .
   .
BEGINTRANSACTION (TRIN) RDS;
   .
   .
   .
MIDTRANSACTION (TRIN, SAVEINPUT) RDS;
   .
   .
   .
ENDTRANSACTION (TRIN, SAVEOUTPUT) RDS;
   .
```

Details regarding the BEGINTRANSACTION, MIDTRANSACTION, and ENDTRANSACTION statements can be found under "DMSII User Language Syntax For Transaction Processing" in this section.

NOTE

A deadlock cannot occur when locking a record if no records are ever locked after it by any transaction.

To illustrate the concept of two-phase transactions and their implications for consistency, consider the following example. Suppose the consistency constraint to be preserved is that data item A of record R1 is equal to data item B of record R2. The following two transaction routines called T1 and T2, written in pseudo-COBOL, preserve the consistency constraint that A=B when run alone.

```
T1:
   LOCK R1.
   COMPUTE A=A+100.
   STORE R1.
   FREE R1.
   LOCK R2.
```

```
      COMPUTE B=B+100.
      STORE R2.
      FREE R2.

   T2:
      LOCK R1.
      COMPUTE A=A*2.
      STORE R1.
      FREE R1.
      LOCK R2.
      COMPUTE B=B*2.
      STORE R2.
      FREE R2.
```

When T1 and T2 are run concurrently, the consistency constraint may or may not be preserved. The following schedule happens to preserve consistency.

```
   T1:                          T2:
      LOCK R1.
      COMPUTE A=A+100.
      STORE R1.
      FREE R1.
                                   LOCK R1.
                                   COMPUTE A=A*2.
                                   STORE R1.
                                   FREE R1.
      LOCK R2.
      COMPUTE B=B+100.
      STORE R2.
      FREE R2.
                                   LOCK R2.
                                   COMPUTE B=B*2.
                                   STORE R2.
                                   FREE R2.
```

If A=B initially, then if T1 and T2 execute with the following schedule, A is not equal to B when T1 and T2 terminate. (In fact, A is equal to B+100.) Thus, the following schedule destroys consistency.

```
T1:                          T2:
   LOCK R1.
   COMPUTE A=A+100.
   STORE R1.
   FREE R1.
                                LOCK R1.
                                COMPUTE A=A*2.
                                STORE R1.
                                FREE R1.
                                LOCK R2.
                                COMPUTE B=B*2.
                                STORE R2.
                                FREE R2.
   LOCK R2.
   COMPUTE B=B+100.
   STORE R2.
   FREE R2.
```

In the preceding example, T1 and T2 both violate the two-phase condition because FREE R1 is followed by LOCK R2. If FREE R1 is eliminated, then both T1 and T2 satisfy the two-phase condition, and consistency can always be preserved. If R1 and R2 are records from the same data set, then the data set must be invoked twice to eliminate the FREE R1.

The programmer can simply code many transaction update routines to lock all the records that the routine updates or examines, perform the appropriate update, and then free all records implicitly at ENDTRANSACTION. (The freeing of records is discussed in greater detail under "Reproducibility" in this section.) However, transaction update routines that update or examine many records in a single data set are complicated because a few of the DMSII user language statements execute an implicit FREE statement. In situations where a large number of records are accessed or updated, or both, invoking the data set(s) more than once may not be feasible.

The alternatives to the two-phase condition involve other programmatic conventions. According to the two-phase theorem, if an arbitrary set of transaction routines is two-phase, then multiprogramming them does not cause loss of consistency. However, if programmatic conventions are used that prevent loss of consistency, then the mix of transactions is not arbitrary.

Such programmatic conventions always lock certain control records before retrieving or updating a well-defined collection of certain other protected records. The control records are kept locked while the protected records are accessed or updated or both. This ensures that the protected records can be accessed by only one process at a time. The control record capability can be implemented in two ways.

The first method uses the DMSII user language statement "LOCK TO MODIFY DETAILS". The master record must be locked before any of its details, or any of the details of its details, can be updated. Each master record is a control record, and a master record and all its descendants are the protected records. For this method, the programmatic convention is enforceable by DMSII, but applies only to embedded structures.

The second method is more generally applicable: to lock a certain disjoint data set record as the first operation in a transaction update routine. The disjoint record could serve, for example, as the control record for a particular data set, a set of data sets, or a set of records in a single data set. If every update routine accesses protected records only through the control record, then the two-phase convention is satisfied. However, the choice as to the number of records controlled by a particular control record can influence the potential throughput of the data base.

When control records are used, the two-phase convention can be paraphrased as follows. During the first part of the update routine, control records are locked and no records are freed. The protected records can then be accessed after the last control record is locked and until the first control record is freed. Once the update routine is complete, the control records can be freed at ENDTRANSACTION.

REPRODUCIBILITY

DMSII is designed to maximize total throughput by allowing more than one update program to execute simultaneously. When multiprogramming is used, the issue of reproducibility of results arises. Due to random timing behavior, the interaction of transactions, and other factors, reproducibility is difficult to achieve. The following are some of the complications involved:

1. If a backup dump of the data base must be loaded to reprocess transactions, the results of the reprocessed transactions are usually not the same.

2. If the data base was updated through the use of multiprogramming, the previous results cannot be reproduced unless reproducibility was designed into the user-written program from the beginning.

3. DMSII RECOVERY backs out a number of transactions, which must then be reprocessed before new transactions are processed. Achieving reproducibility of previous results while reprocessing backed-out transactions requires special handling.

The midtransaction phase (described under "Data Base Consistency And Update Library Programming" in this section) provides the key to achieving reproducibility. Concurrently executing two-phase transactions that are in midtransaction phase are independent of each other because they have disjoint collections of records locked. Transactions using a programmatic conventions equivalent to two-phase have disjoint collections of control records locked and do not share any protected records.

The independence of transactions means that a serial schedule for executing these concurrent transactions can easily be constructed. Such a schedule is equivalent to the original multiprogrammed schedule. In other words, transactions can be executed in any order whatsoever because the transactions update disjoint parts of the data base and do not interact.

For example, consider two concurrently executing transactions, A and B. Suppose A is in midtransaction phase, and B is in the first phase (when records are being locked but not freed), waiting on a record that A has locked. In general, the results of B depend on the results of A. However, B cannot enter midtransaction phase until A has completed midtransaction phase. Due to the effectiveness of the locking strategy, if A is run alone followed by B, the results are the same as if these transactions had executed concurrently. Therefore, the order in which

transactions enter midtransaction state determines the order in which the transactions can be executed in an equivalent serial schedule.

In TPS an Update Library transaction processing routine executes the DMSII user language MIDTRANSACTION statement when the routine is in midtransaction phase. Among other things, this causes a procedure in the Transaction Library to be called that saves the input transaction record in a TRHISTORY transaction journal data file. If the input transaction records in a TRHISTORY data file are reprocessed serially, then the results obtained when the data base was originally updated are reproduced.

Errors can cause additional complications in regard to reproducibility. Suppose that a program error in a transaction update routine, or an error in the data of an input transaction record, appears such that when the record is processed, a fault always occurs, causing an ABORT RECOVERY. When this transaction record is reprocessed after transactions are backed out, an ABORT occurs again. In order to avoid an infinite loop, the Transaction Library recognizes that a particular transaction record always causes an ABORT and rejects it.

This rejection by TPS introduces a new reproducibility problem, because other transactions may have depended on some records that the erroneous transaction had freed during its second phase. Because the erroneous transaction is ultimately discarded, the results of reprocessing these other transactions may be different. This situation is undesirable because output may already have been sent to remote terminals. This problem is referred to as the terminal reproducibility problem.

The problem of terminal reproducibility has two possible solutions. The first solution is to require that all transactions use the SYNC option in the ENDTRANSACTION statement to protect themselves from erroneous transactions. This solution may be undesirable because it increases overhead and decreases potential throughput.

The alternative solution is to keep all records (or, in the case of programmatic conventions equivalent to two-phase, all control records) locked until the transaction has completed, and to let the ENDTRANSACTION statement free them. Once ENDTRANSACTION has been executed, the transaction can no longer cause an ABORT. The records it has modified cannot be accessed until execution of ENDTRANSACTION is complete. This convention is considered the preferred solution to the terminal reproducibility problem.

In summary, a programmatic convention equivalent to two-phase should be used in programming the Update Library, and all records should be kept locked until freed by an END-TRANSACTION statement. A FIND statement

cannot be used, except on protected records. The implicit FREE statement performed by the FIND, LOCK, and DELETE statements must be carefully handled.

To minimize the amount of time spent in transaction state, as many records as possible should be locked outside of transaction state. If no records are locked inside transaction state, BEGINTRANSACTION is the last statement that can return a deadlock exception. This makes deadlock handling as simple as possible. The transaction update routine should be coded to branch back to the beginning of the transaction, because the data base could not have been changed outside of transaction state.

7 REMOTE LIBRARY AND HOSTINTERFACE

This section discusses remote real-time updating of the data base through the use of the Remote Library and the HOSTINTERFACE program. Error conditions and communication interruptions that apply to remote system/host system communications are also explained.

COMPILING THE REMOTE LIBRARY

The Remote Library and the HOSTINTERFACE program can be compiled automatically by SYSTEM/TFL whenever a new TRDESCRIPTION file is created. Automatic compilation occurs only if the TFL ZIP dollar-card option is set. The title of the resulting Remote Library code file is <base name>/CODE/REMOTELIB, and the title of the resulting HOSTINTERFACE code file is <base name>/CODE/HOSTINTERFACE.

Alternatively, the user may compile the Remote Library by compiling the source files TRBASE/REMOTELIB, TRBASE/PROPERTIES, and the TRDESCRIPTION file with the DMALGOL compiler, as shown in Figure 7-1. All required processor options are included in the TRBASE/REMOTELIB source file. The internal DMALGOL compiler file named DASDL must be label-equated to the appropriate TRDESCRIPTION file, named <base name>/TRDESCRIPTION.

The user may compile the HOSTINTERFACE program by compiling the source files TRBASE/HOSTINTERFACE and TRBASE/PROPERTIES and the TRDESCRIPTION file with the DMALGOL compiler, as shown in Figure 7-2. All required processor options are included in the TRBASE/HOSTINTERFACE source file. The internal DMALGOL compiler file named DESCRIPTION must be label-equated to the appropriate TRDESCRIPTION file, named <base name>/TRDESCRIPTION.

A compile-time check ensures that the TRBASE/PROPERTIES property level matches the property level contained within the TRDESCRIPTION file. If the level of the TRBASE/PROPERTIES file exceeds the level of the TRDESCRIPTION file, the syntax error "DESCRIPTION FILE TOO OLD FOR PROPERTIES" is issued. If the level of the TRDESCRIPTION file exceeds the level of the TRBASE/PROPERTIES file, the syntax error "PROPERTIES TOO OLD FOR DESCRIPTION FILE" is issued.

Examples:

The following examples show two WFL jobs for compiling the HOSTINTERFACE program and the Remote Library:

A Series DMSII Transaction Processing System (TPS)

PCN 1164043-001 Release 3.6 December 1985

```
?JOB COMPILEHOSTINTERFACE;
USER SITE;
FAMILY DISK= TPS OTHERWISE DMS;
COMPILE BANKTR/CODE/HOSTINTERFACE
        WITH DMALGOL [CT] LIBRARY;
OPTION=FAULT DSED ARRAYS;
SUBSYSTEM := BLUE;
PL SUBSPACES=0;
PL FILE DASDL = BANKTR/TRDESCRIPTION;
PL FILE TAPE=TRBASE/HOSTINTERFACE;
PL FILE ERRORFILE(KIND=DISK,PROTECTION=SAVE,TITLE=ERROR/HI);
DATA CARD;
$MERGE LINEINFO;
IF CT ISNT COMPILEDOK THEN
BEGIN
      ABORT "*** SYNTAX ERRORS ***";
END;
?END JOB;


?JOB COMPILEREMOTELIB;
USER SITE;
FAMILY DISK= TPS OTHERWISE DMS;
COMPILE BANKTR/CODE/REMOTELIB
        WITH DMALGOL [CT] LIBRARY;
OPTION=FAULT DSED ARRAYS;
SUBSYSTEM := BLUE;
PL SUBSPACES=0;
PL FILE DASDL = BANKTR/TRDESCRIPTION;
PL FILE TAPE=TRBASE/REMOTELIB;
PL FILE ERRORFILE(KIND=DISK,PROTECTION=SAVE,TITLE=ERROR/RL);
DATA CARD;
$MERGE LINEINFO;
IF CT ISNT COMPILEDOK THEN
BEGIN
      ABORT "*** SYNTAX ERRORS ***";
END;
?END JOB;
```

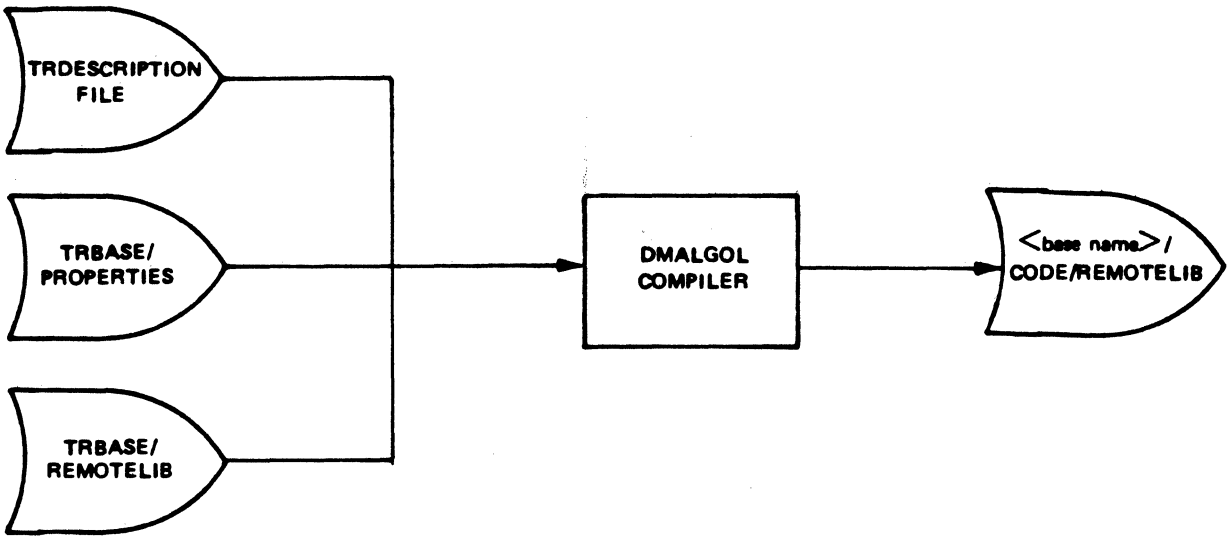


Figure 7-1. Compilation of the Remote Library

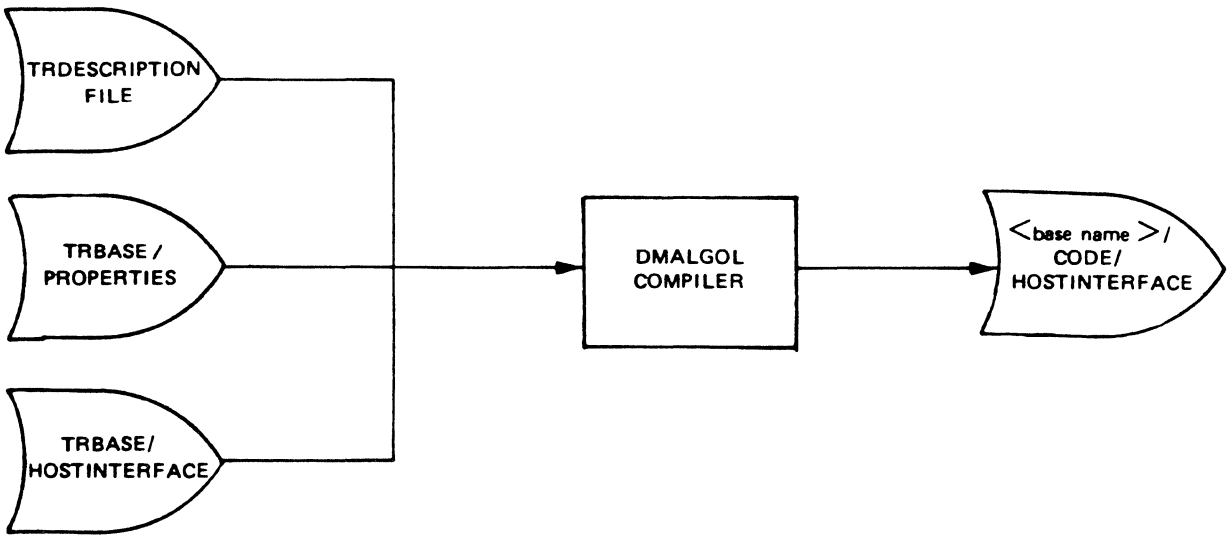


Figure 7-2. Compilation of the HOSTINTERFACE Program

HOW THE REMOTE LIBRARY WORKS

TPS enables the user to establish a data base on one system and to run user-written programs that access the same data base on other systems in the user's network. TPS provides an efficient method for interfacing a DMSII program executing on a remote system with the ACCESSROUTINES and the data base on a host system.

TPS includes two software modules named the Remote Library and the HOSTINTERFACE program, which enable data base processing from a remote system. The Remote Library runs on the remote system with the user programs. It is analogous to the Transaction Library that runs on the host system, and it utilizes the same set of library entry points (which are discussed in Section 5 of this manual).

The Remote Library is compiled on the host system and must be copied over to the remote system, and can be copied under the name <base name>/CODE/HOSTLIB. Thus, a program on the remote system calls the same

entry points that a program on the host system calls, except that the program on the remote system is actually calling the Remote Library. In this manner, user-written programs can be executed on the host system or on any remote system in the user's network without having to be rewritten. This capability promotes the portability of user-written software.

When a user-written program calls the Remote Library for the first time, a library stack is initiated that is shared by all users of the transaction base. One private library stack is also initiated for each invocation of the remote library. (A sample Remote Library stack configuration is given at the end of this section.) The private Remote Library stack initiates the HOSTINTERFACE program as a dependent process on the host system. One HOSTINTERFACE program is initiated for each private Remote Library stack on a remote system.

The Remote Library and HOSTINTERFACE modules communicate through port files that are declared and opened in each program. When a user-written program calls a Remote Library entry point, the Remote Library informs the HOSTINTERFACE which entry point the user-written program called and passes along the parameters required by that entry point. The HOSTINTERFACE, which has declared each of the Transaction Library entry points, simply calls the appropriate entry point with the correct parameters. Any output resulting from the Transaction Library entry point call is sent by the HOSTINTERFACE to the Remote Library. The Remote Library, in turn, returns the results to the user-written program.

Thus, the entry point call that the user-written program made on the remote system is actually made by the HOSTINTERFACE on the host system. The net result is that the correct entry point in the Transaction Library is called. This call then accesses the Update Library, which in turn accesses the data base, and the result of the transaction is received in the user-written program.

Additional overhead is required to pass information back and forth through the port files. However, aside from a slower response, the user-written program functions the same as if it were in direct communication with the Transaction Library and the data base.

ERROR CONDITIONS

Certain remote-host error results may be returned to a user-written program after it calls on the Remote Library. For this reason, user-written programs should be written to handle these errors if they will ever be run on a remote system, even though these programs may be run on the host system most of the time.

The Remote Library recognizes the HOSTNAME of the system on which the data base is located, because it is specified in TFL. If the identifier given in TFL is not the HOSTNAME of a system recognized by the remote system, the Remote Library returns an error result of 100.

If the Remote Library loses communication with the HOSTINTERFACE, because of either a data communications problem or a failure of the host system, the Remote Library cannot continue.

If the user specifies a value for the TIMEOUT parameter of OPENTRBASE, then the Remote Library waits the specified number of seconds for the host system to come back up. If the value for TIMEOUT is 0, or if the other system does not come up, the program receives a 101 error result indicating that the host system is down. If the host system comes back up, a 102 error result is returned to indicate that the program can re-try the transaction because the host system is available.

The user-written program running on the remote system must handle these errors and resubmit the last input, if necessary, just as though the user-written program were restarting after a Halt/Load on its own system. The RETURNRESTARTINFO and RETURNLASTRESPONSE Remote Library entry points should be used to determine where to restart.

The user-written program that receives a 101 error result must first determine whether the host system is running before it can restart. The program could be coded to periodically call an entry point in the Remote Library and receive 101 results until the host system comes up. Otherwise, the program could terminate when the 101 error result is returned, and the program could be restarted when the host system is brought back up.

A error in the port file mechanism returns a 103 result. This value indicates that the response received from the HOSTINTERFACE program is not the correct response transaction for the input transaction that was submitted.

If the remote system goes down, the HOSTINTERFACE program terminates. When the remote system is restarted, the Remote Library initiates another HOSTINTERFACE, and processing can continue.

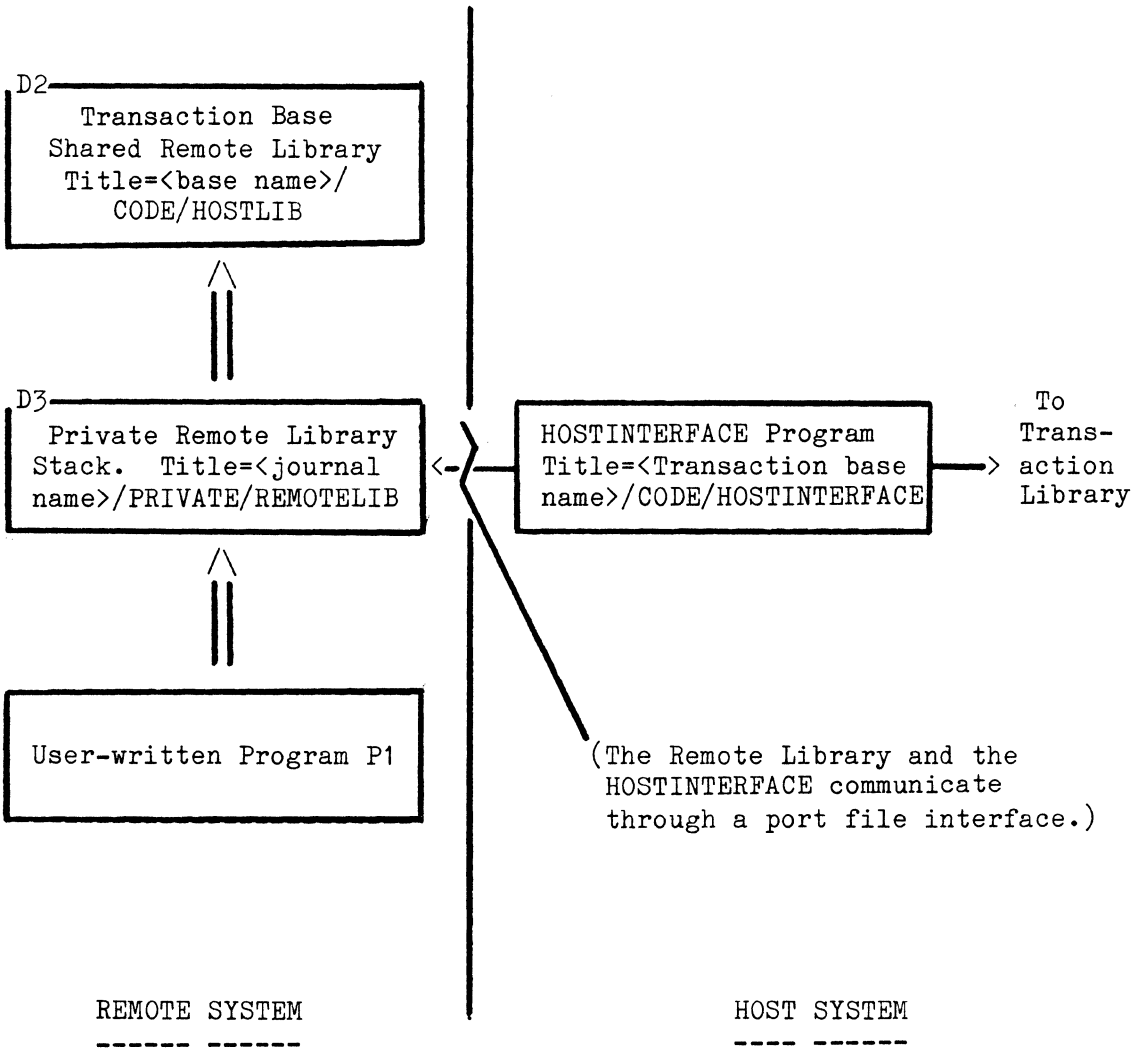
When the user-written program calling the Remote Library has finished processing, the Remote Library informs the HOSTINTERFACE and both the Remote Library and HOSTINTERFACE terminate normally.

SAMPLE REMOTE LIBRARY STACK CONFIGURATION

The following diagram shows a typical stack configuration for the Remote Library. In this example, a user-written program named P1 is invoking the TRHISTORY journal and has just completed a call on the library entry point OPENTRBASE.

NOTE

For this example, the title of the Remote Library is <transaction base name>/CODE/REMOTELIB if the Remote Library is not copied from the host system to the remote system as <base name>/CODE/HOSTLIB. This example assumes that <base name>/CODE/REMOTELIB has been copied as <base name>/CODE/HOSTLIB.


D2
Transaction Base
Shared Remote Library
Title=<base name>/
CODE/HOSTLIB
D3
Private Remote Library
Stack. Title=<journal
name>/PRIVATE/REMOTELIB
HOSTINTERFACE Program
Title=<Transaction base
name>/CODE/HOSTINTERFACE
To
Trans-
action
Library
User-written Program P1
(The Remote Library and the
HOSTINTERFACE communicate
through a port file interface.)
REMOTE SYSTEM
HOST SYSTEM

8 EXCEPTION HANDLING

EXCEPTION REPORTING

The Transaction Library entry points return values. If the value returned is 0 (zero), then the entry point is performed as expected. Non-zero values are called "exceptions". Exceptions are returned to user programs any time the Transaction Library cannot successfully or correctly complete a call on an entry point. This is called an "exception situation." Exceptions and their meanings are listed later in this section.

As a general rule, an exception situation only affects the user program that receives the exception; other user programs are unaffected. There are two kinds of exception situations that violate the general rule. One kind is a fatal error detected by the Transaction Library. The other kind is the result of an ABORT caused by some program.

A fatal error occurs when the Transaction Library encounters a situation that halts further processing. In a fatal error exception situation, the Transaction Library must be brought down. For the program that detected the fatal error, an operator must respond to an ACCEPT statement which will terminate the program. While the Transaction Library is still up, any user program that invokes any entry point during a fatal error exception situation gets back an immediate exception value of 313 (TRBASEDOWN - nothing was done in the entry point). When the user program is terminated, locks held by user programs are not released by the Transaction Library. Fatal errors are listed later in this section.

In the case of an ABORT exception situation, there is no one single exception value returned to a user program that caused it. Other user programs in the act of processing transactions via some entry point call may be momentarily stopped by the Transaction Library. The Transaction Library will automatically handle the reprocessing of backed out transactions. After the Transaction Library has handled the ABORT exception situation, the user programs that were stopped then resume normal processing.

Fatal Errors

The following paragraphs describe the fatal errors returned by Transaction Library entry points. Where appropriate, actions to prevent the errors from recurring are also given.

MUSTHAVELOCK (Value = 1)

Cause: The running stack did not own a lock that it must have.

Action: Bringing down the transaction base and restarting it again may fix the problem.

MUSTNOTHAVELOCK (Value = 2)

Cause: The running stack owned a lock that it must not have.

Action: Bringing down the transaction base and restarting it again may fix the problem.

CFOPENERROR (Value = 3)

Cause: The Transaction Library was unable to open the journal control file, or the journal control file was too old for the Transaction Library.

Action: If the journal control file is too old, load the correct control file. If no valid control file can be found, run TRUTILITY to recover the control file (Refer to Section 11 of this manual for more information concerning TRUTILITY.)

FLUSHCFERROR (Value = 4)

Cause: The Transaction Library was unable to update the journal control file because an I/O error occurred.

CLOSECFERROR (Value = 5)

Cause: The Transaction Library was unable to close the journal control file.

CANNOTCREATENEXTTRFILE (Value = 6)

Cause: An error occurred when the Transaction Library tried to create a new data file.

CANNOTRECOVERLASTBLOCK (Value = 7)

Cause: The Transaction Library was unable to use the last journal data file for recovery.

BADTRBASETIMESTAMP (Value = 8)

Cause: The journal data file time stamp did not match the TRDESCRIPTION file time stamp.

Action: Verify that the journal data file was created with the current version of the TRDESCRIPTION file.

BADTRFILEUPDATELEVEL (Value = 9)

Cause: The journal data file update level did not match the TRDESCRIPTION file update level.

A Series DMSII Transaction Processing System (TPS)

PCN 1164043-001 Release 3.6 December 1985

Action: Verify that the journal data file was created with the current version of the TRDESCRIPTION file.

BADTRFILEVERSION (Value = 10)

Cause: The journal data file format level was greater than that expected by the Transaction Library.

Action: Verify that the Transaction Library represents the most current version of the software.

BADTRFILESERIALNUM (Value = 11)

Cause: The block serial number of the first block of a journal data file was less than 1. This value implies that the data file is corrupted.

CWMISMATCH (Value = 12)

Cause: The left control word of a transaction record did not match its right control word.

UNEXPECTEDEOF (Value = 13)

Cause: The Transaction Library encountered an unexpected end-of-file situation.

TAPENOTIMPLEMENTED (Value = 15)

Cause: The TRDESCRIPTION file may be corrupted.

Action: Submit the TFL source input to the TFL processor to generate a new description file.

WRONGDIRECTION (Value = 16)

Cause: The Transaction Library tried to read a transaction journal data file backwards.

BADSERIALNUMTOREAD (Value = 18)
CURABSNUNDEFINED (Value = 19)

Cause: The Transaction Library was requested to read a block that had an invalid serial number.

READIOERROR (Value = 20)
READCHECKSUMERROR (Value = 21)
READBLKSZERROR (Value = 22)
READABSNERROR (Value = 23)
READBADTIMESTAMP (Value = 24)

Cause: The data block read in by the Transaction Library was either corrupted or simply could not be read.

FAILUREINREJECTTRTRANSACTION (Value = 27)

Cause: An I/O or hardware error occurred when the Transaction Library tried to reject a bad input transaction record.

FAILUREINRERUN (Value = 28)

Cause: The Transaction Library received an exception while reprocessing transactions. This exception could occur for a number of reasons, such as the occurence of a fault in the Update Library.

UNKNOWNTRSTATE (Value = 29)

Cause: The TRSTATE control item field of a transaction record was not recognized by the Transaction Library.

NOTOPENOUTPUT (Value = 30)

Cause: The Transaction Library attempted to initiate a write for a non-update user.

DATAFILEATTERR (Value = 32)

Cause: The Transaction Library was unable to set up the journal data file attributes.

In addition to the above fatal errors, if a data base was invoked in the user's transaction base, the Transaction Library will terminate abnormally with the following fatal errors if the Transaction Library was not able to communicate with the ACCESSROUTINES. The communication is done through DMINQ functions provided by DMALGOL.

Fatal Mnemonic	Value
FAILUREINRESETTINGTRADDRESS	37
FAILUREINSETTINGTIMESTAMP	38
FAILUREINGETLASTTRADDRESS	39
FAILUREINABORTFINISHED	40
FAILUREINRESETTINGTPSCLOSEDFLAG	41
FAILUREINRESETTINGTPSABORTPENDINGFLAG	42

Cause: There may have been a fault in the ACCESSROUTINES just prior to the DMINQ function call.

FAULTINTPSCODE (Value = 43)

Cause: A fault in TPS code while excuting an entry point.

Action: Fault handling code will display the stack number of the task that got the fault, the fault history and fault number for subsequent analysis.

Exceptions

The following list shows the values returned for exceptions by the Transaction Library entry points. For each value, the corresponding exception is described.

CATEGORY 1 : 001 - 099 (ATTENTION)

Value	Meaning
1	The transaction system encountered an end-of-file situation when reading a journal data file. In this situation, no transaction record is returned.
2	The transaction record returned from the Transaction Library to the user program was a rejected transaction record. In this situation, the user program cannot rely on this transaction record for further operation.
3	No input or restart transaction record could be found for a user. This situation can occur if RETURNRESTARTINFO is called when no record has been processed or tanked by this user. In this situation, no restart information exists for the user.
4	No response transaction record could be found for the user. This situation can occur if RETURNRESTARTINFO is called by a tank user or if no response transaction record exists for an update user.
5	The transaction record address returned to the RETURNLASTADDRESS call referenced a restart record. This result is for users' reference only.
6	Similar to Category 1 Exception 4, except that no input transaction record could be found for a user.
33	An attempt was made to process an input transaction record before the Transaction Library reprocessed transactions backed-out by DMSII Recovery. After the Transaction Library

reprocesses the backed-out transactions, the input transaction is automatically processed.

34 The Transaction Library attempted to open the next file in sequence, but the file was not present.

36 An I/O error occurred when the Transaction Library tried to read a transaction record from a file.

CATEGORY 2 : 100 - 199 (REMOTE-HOST INTERFACE)

Value	Meaning

100 The name of the host system, which was specified in the TFL source, was not the HOSTNAME of a system recognized by the Remote System.

101 The host system could not be reached from the remote system. This exception can occur if a BNA link does not exist or if the host system is not running, after waiting the specified number of seconds for the system to come back up. (For more information, refer to the TIMEOUT parameter of the OPENTRBASE Transaction Library entry point in Section 5 of this manual.

To correct the problem, bring up either the host system or the BNA link.

102 The host system has come back up. This result is returned to indicate that the program can re-try the transaction because the Host System is now available.

103 The TRCOUNTER of a transaction record did not match the remote system counter. This mismatch implies that the output transaction record was not valid for the imput transaction record sent from the remote system.

CATEGORY 3 : 200 - 209 (BAD PARAMETER VALUES)

Value	Meaning

200 The USEROPTION parameter for OPENTRBASE was greater than 5 or less than 1.

To correct the problem, change the parameter value to a value within the correct range.

201 The user ID string parameter was unknown to the transaction journal. This exception can occur if LOGONTRUSER or TRUSERIDSTRING is called without calling CREATETRUSER first. It can also occur if a bad parameter is passed to LOGONTRUSER or TRUSERIDSTRING.

To correct the problem, make sure that the parameter is correct and that CREATETRUSER has been called before calling LOGONTRUSER or TRUSERIDSTRING.

Value	Meaning

202 The user identified by this ID number was not logged on (that is, not marked active). This exception can occur if any entry points except CREATETRUSER are called before calling LOGONTRUSER.

To correct the problem, call LOGONTRUSER to log on the user before calling other entry points.

204 The user ID string parameter to CREATETRUSER was not unique to the particular transaction journal. In other words, an existing user had the same ID string as the parameter.

To correct the problem, either purge the existing user by calling PURGETRUSER or assign a new ID string parameter to CREATETRUSER.

205 The user ID string parameter to LOGONTRUSER was already marked active.

To correct the problem, either assign another ID string parameter to LOGONTRUSER or continue to call other entry points because the user is already logged on.

206 The file number parameter for SEEKTRANSACTION was greater than 9999 or less than 1.

To correct the problem, make sure the file number parameter is within the allowable range.

207 The size of an output transaction record parameter of a Transaction Library entry point was not large enough to hold a transaction record returned from the Update Library.

To correct the problem, make sure that the largest transaction record format invoked in the user program is larger than the largest transaction record format returned from the Update Library.

208 The <parameters> option passed to the HANDLESTATISTICS entry point is not valid.

209 The STATISTICS option has not been set in the TFL source file.

CATEGORY 4 : 300 - 399 (FUNCTION NOT PERMITTED)

Value	Meaning

301 An attempt was made to open the transaction base when it was already open. This exception can occur if OPENTRBASE is called more than once in the same user-written program. OPENTRBASE can be called only once in each user-written program, and it must be the first entry point called.

To correct the problem, make sure that OPENTRBASE is the first entry point called and that it is called only once within the program.

302 The Transaction Library could not process any transaction because the transaction base was not open for tanking. This exception can occur if PROCESSTRNORESTART, PROCESSTRANSACTION or PROCESSTRFROMTANK is called when the transaction base is open for tanking.

To correct the problem, the user should never try to process a transaction record when the transaction base is open for tanking.

303 The Transaction Library could not tank any transaction record because the transaction base was open for updating. This exception can occur if TANKTRANSACTION or TANKTRNORESTART is called to tank to the TRHISTORY journal. TRHISTORY is a reserved word for TPS, and the journal named TRHISTORY can be used only for updating.

To correct the problem, do not try to tank a transaction record when the transaction base is open for updating. In addition, do not try to give the name TRHISTORY to the tank journal.

305 The transaction base was not open. This exception can occur if other entry points are called before OPENTRBASE.

To correct the problem, always call OPENTRBASE before calling other Transaction Library entry points.

306 The transaction base was not open for reading. This exception can occur if READTRANSACTION is called when the transaction base is open for updating or tanking.

To correct the problem, do not call READTRANSACTION when the transaction base is open for updating or tanking.

Value Meaning

307 The transaction base was not open for writing. This exception can occur if entry points are called to tank or process a transaction record when the transaction base is open for reading.

To correct the problem, do not tank or process any transaction records when the transaction base is open for reading.

308 The transaction base was not open for reading. This exception can occur if SEEKTRANSACTION is called when the transaction base is not open for reading.

To correct the problem, do not call SEEKTRANSACTION when the transaction base is not open for reading.

309 No response transaction record was returned for a tanker. This exception can occur if RETURNLASTRESPONSE is called when the transaction base is open for tanking.

To correct the problem, do not call RETURNLASTRESPONSE when the transaction base is open for tanking.

310 The Transaction Library attempted to update a tank journal. This exception can occur if an attempt is made to open a transaction base for updating, specifying a tank journal's name rather than TRHISTORY.

To correct the problem, use the TRHISTORY journal when the transaction base is open for updating.

Value	Meaning
311	The user attempted to inquire against a tank journal. This exception can occur if an attempt is made to open a transaction base for inquiry, specifying a tank journal's name rather than TRHISTORY. To correct the problem, use the TRHISTORY journal when the transaction base is open for inquiry.
313	A fatal error occurred before an entry point in the Transaction Library was called. The transaction base is unusable after a fatal error. The program that receives this exception should terminate.
314	The exclusive update was not allowed because the transaction base was already in use. This exception can occur if OPENTRBASE is called for exclusive update when the transaction base is already open for updating.
315	An attempt to open the transaction base for update was not allowed because the transaction base was already open for exclusive update. This exception can occur if OPENTRBASE is called for update when the transaction base is already open for exclusive update.

CATEGORY 5 : 400 - 499 (TRANSACTION REJECTED)

NOTE

The exception number for a restart record is the exception number of its input record plus 50. The exception number for a response record is the exception number of its input record plus 150. For example, 401 indicates a bad timestamp for an input record, 451 indicates a bad timestamp for the restart record, and 551 indicates a bad timestamp for the response record.

Value	Meaning
400	The input transaction record was not created. This exception can occur if a transaction record is passed to TANKTRANSACTION, TANKTRNORESTART, PROCESSTRANSACTION, PROCESSTRFROMTANK, or PROCESSTRNORESTART without first using a CREATE statement to initialize the transaction record. To correct the problem, initialize the transaction record with a CREATE statement before using it as a parameter.
401 451 551	The transaction record time stamp did not match the creation date time stamp of the TRDESCRIPTION file.
402 452 552	The update level of the transaction record format associated with the transaction record was greater than the maximum update level of the TRDESCRIPTION file against which the Transaction Library was compiled, or less than the base update level of the TRDESCRIPTION file against which the Transaction Library was compiled. This exception can occur in the following situations: 1. After a TFL update, if a user-written program is run to process a transaction record with an old format. 2. If the Transaction Library is compiled with an old version of the TRDESCRIPTION file. To correct the problem, either recompile the user-written program or use the correct version of the TRDESCRIPTION file to compile the Transaction Library.

Value	Meaning
403 433 553	The transaction record update time stamp was not valid (with respect to the TRDESCRIPTION file) for the corresponding transaction record update level.
404 454 554	The transaction record format associated with a transaction record was not known to the transaction base. This exception can occur if an old user-written program is not recompiled after a TFL update. To correct the problem, recompile the user-written program.

405
455
555 The format VERIFY Boolean expression was evaluated to FALSE.

406
456
556 The transaction subformat associated with the transaction record was not known to the transaction base. This exception can occur if an old user-written program is not recompiled after a TFL update.

To correct the problem, recompile the user-written program.

407
457
557 The subformat VERIFY Boolean expression was evaluated to FALSE.

408
458
558 The transaction subbase associated with a transaction record was not known to the transaction base. This exception can occur if an old user-written program not recompiled after a TFL update.

To correct the problem, recompile the user-written program.

409
459
559 The transaction record was not accepted by the Transaction Library because the usercode, the name of the program that submitted the transaction, or both did not pass the GUARDFILE check for the subbase containing the transaction word.

CATEGORY 6 : 500 - 599 (TRANSACTION FAILED)

Value	Meaning
501	The transaction record failed after BEGINTRANSACTION within the Update Library.
502	The transaction record failed after the input transaction record was saved in the journal data file.
503	The transaction record failed after MIDTRANSACTION within the Update Library.

504 The transaction record failed after the response transaction record was saved in the journal control file.

505 A transaction record failed during reprocessing.

506 No MIDTRANSACTION statement was encountered after the BEGINTRANSACTION statement in the Update Library.

507 No ENDTRANSACTION statement was encountered after the MIDTRANSACTION statement in the Update Library.

CATEGORY 7 : 900 - 999 (SOFTWARE/HARDWARE/PROCEDURAL ERRORS)

Value	Meaning

901 More than 128 users attempted to process against the same journal, or more than 128 journals attempted to access the transaction base.

902 An I/O error occurred when the Transaction Library tried to read or update the journal control file.

904 The Transaction Library could not expand a compacted transaction record.

905 The Transaction Library could not compact an expended transaction record.

906 Reserved for system use.

907 The last user number allocated in the control file did not match the last user number in the USERINFO SYSTEMTR subformat.

908 The user ID string could not be located within an internal array during a call to LOGONTRUSER.

909 An I/O error occurred in SEEKTRANSACTION.

910 The last good transaction record address stored in the data base control file was not valid with respect to the journal data file when the Transaction Library was attempting to reprocess transactions.

911 The tank journal control file was out of sync with the data file.

To correct the problem, run TRUTILITY to recover the tank journal control file.

912 The version of the journal control file being used was not the most current version.

To correct the problem, load the right version of the journal control file.

DMS EXCEPTION HANDLING

The Transaction Library processes each transaction record through the Update Library, which contains all the data base management statements that actually perform update and inquiry actions against the data base. The Update Library handles most exceptions to data base management statements. However, the Transaction Library handles exceptions related to ABORT detection and program restarting because it reprocesses transactions backed out during an ABORT.

The exceptions returned to Update Library transaction processing routines are the same exceptions returned from any other DMSII routines, except for certain differences related to the OPEN, BEGINTRANSACTION, MIDTRANSACTION and ENDTRANSACTION statements. For complete descriptions of all DMSII exceptions, refer to Appendix D in the DMSII User Language Interface Software Operation Guide. The exceptions discussed in the following paragraphs can be returned for the indicated data base management statements in the Update Library.

Returned from OPEN:

OPENERROR

(All previous values of OPENERROR apply, plus two new values.)

Value	Meaning
45	This open option is not permitted while TPS is active. A program tried to perform a regular OPEN UPDATE while another program was active with OPEN TRUPDATE.
46	This open option is not permitted until TPS recovery has completed. This situation resulted because an ABORT or Halt/Load occurred, and TPS had not yet reprocessed the backed-out transactions. A regular OPEN UPDATE is not allowed until the transaction base is brought up again by any TPS user-written program. When the transaction base is brought up again, TPS recovery runs and reprocesses the backed-out transactions.

Returned from BEGINTRANSACTION:

ABORT

Value	Meaning
1	The normal ABORT exception that indicates a program should restart. This exception does not occur in a program using the TPS form of BEGINTRANSACTION.
2	An ABORT occurred, and the Transaction Library has not yet begun reprocessing transactions. The current transaction should exit the Update Library. The Transaction Library resubmits this transaction after it reprocesses the backed-out transactions.

DEADLOCK

Value	Meaning
1	A deadly embrace occurred during an attempt to lock a record. The system will have automatically freed all records for this program.
2	A time-out occurred during an attempt to lock a record.

AUDITERROR

Value	Meaning
1	A BEGINTRANSACTION statement was attempted when the data base was already in transaction state.
4	A normal BEGINTRANSACTION statement was attempted on a data base that was opened by TPS.

5	A TPS BEGINTRANSACTION statement was attempted on a data base that was not opened by TPS.
6	The transaction record had an improper TRSTATE value. That is, the state of the transaction record was unexpected and indicated a system software problem.

DATAERROR

Value	Meaning
1	Some REQUIRED items were null, or the VERIFY condition failed.

Returned from MIDTRANSACTION:

AUDITERROR

Value	Meaning
3	A MIDTRANSACTION statement was attempted when the data base was not in transaction state.
6	The transaction record had an improper TRSTATE value. That is, the state of the transaction record was unexpected and indicated a system software problem.

Value	Meaning
7	A MIDTRANSACTION statement was attempted on a data base that was not opened by TPS.

Returned from ENDTRANSACTION:

ABORT

Value	Meaning
1	The normal ABORT exception that indicates a program should restart. This exception does not occur in a program using the TPS form of BEGINTRANSACTION.

DEADLOCK

Value	Meaning
1	A deadly embrace occurred during an attempt to lock a record. The system will have automatically freed all records for this program.
2	A time-out occurred during an attempt to lock a record.

AUDITERROR

Value	Meaning
2	An ENDTRANSACTION statement was attempted when the data base was not in transaction state.
4	A normal ENDTRANSACTION statement was attempted on a data base that was not opened by TPS.
5	A TPS ENDTRANSACTION statement was attempted on a data base that was not opened by TPS.
6	The transaction record had an improper TRSTATE value, indicating a system software problem.

DATAERROR

Value	Meaning
1	Some REQUIRED items were null, or the VERIFY condition failed.

EXCEPTION HANDLING MECHANISM

The Update Library should be coded to capture exceptions that might result from each statement executed against the data base. Depending on the value of the result, the program must decide whether to take appropriate action or exit the Update Library. If the program exits the Update Library before performing a successful BEGINTRANSACTION statement, the Transaction Library notices and returns a 505 exception result to the program (refer to CATEGORY 6 in this section). If a transaction fails after BEGINTRANSACTION, a 501, 502, 503, or 504 exception result is returned to the program and an ABORT is caused.

For DMSII exception results returned during transaction state (including DMSII exception results returned on a BEGINTRANSACTION, MIDTRANSACTION, and ENDTRANSACTION), the Update Library should be exited. The TPS application should examine the TPS exception result and should take appropriate action. See the Update Library programming examples in Appendix C.

An inquiry transaction can still optionally save a response transaction before returning to the Transaction Library. Such a transaction is saved by explicitly calling the SAVERESPONSETR entry point.

If the Update Library does not capture exceptions, the ACCESSROUTINES cause the user-written program to be terminated abnormally. After the program terminates, the private Transaction Library, Update Library, and Structure Information Block (SIB) all disappear.

If the Update Library is in transaction state when the exception is encountered, an ABORT is caused. Because the user-written program is terminated abnormally, no useful information can be returned about the cause of the ABORT.

9. SECURITY

TPS provides security through the use of guardfiles. With guardfiles, the use of transaction record types such as formats and subformats can be restricted to particular programs, to particular usercodes, or both. Guardfiles can be created using the SYSTEM/GUARDFILE utility described in the A Series System Software Utilities Reference Manual.

A guardfile can be associated with a transaction subbase in TFL, just as a guardfile can be associated with a logical data base in DASDL. The guardfile restricts access to transaction types that have been logically grouped together. If the guardfile is stored under a usercode, that usercode must be specified in the TFL <transaction subbase> definition (described in Section 3 of this manual). If the guardfile is stored as a system file, the asterisk (*) must be specified.

Although the guardfile security provided for TFL overlaps the guardfile security provided in DASDL, it can be used independently of, or in conjunction with, DMSII security. For example, a user might restrict access to certain transaction records independently of whether logical databases are declared in the DASDL for the corresponding data base.

The Transaction Library checks the transaction records against the guardfiles as transaction records enter the Library for processing. Rather than checking every record from a user, the Transaction Library checks the guardfile for each subbase the first time a transaction within that subbase is processed. For future references, the Transaction Library remembers whether the transaction user was a valid user.

The subbase number is placed in the control portion of each transaction record when it is initialized. A record without a subbase number is not properly initialized and is rejected by the Transaction Library. If an unauthorized usercode or program attempts to process a transaction that it is not allowed to access, the Transaction Library rejects the request and returns an appropriate result to the program.

Example:

This sample subbase defined in TFL demonstrates the use of a guardfile attached for security in a banking application.

```
MANAGER TRANSACTION SUBBASE
 (
     CREATEACCT,
     PURGEACCT,
     DEPOSIT,
     WITHDRAWAL,
     STATUS
 ),
     GUARDFILE = BANKTR/MANAGER/GUARDFILE;
```

10. TPS RECOVERY

TPS guarantees the permanence of any transaction that has been processed successfully, provided that certain requirements are satisfied. Thus, the user never needs to reprocess transactions for which changes to the data base have been backed out by a data base RECOVERY operation (for example, after a Halt/Load or ABORT).

DMSII must back out certain transactions during recovery. But those transactions can be automatically reprocessed by TPS. The previous results must be reproduced during this process, so that the user is unaware that the results have been backed out once RECOVERY has completed. After a Halt/Load, the user should be concerned only with restarting the program and with positioning any input or output files that were in use.

TPS utilizes the following basic Recovery operations:

1. Database RECOVERY after an ABORT.

   Performed by DMSII.

2. Database RECOVERY after a Halt/Load.

   Performed by DMSII.

3. Transaction recovery after a Halt/Load.

   Performed by TPS. The end-of-file pointer for the journal currently opened by the program is recovered to avoid losing any information written to the file. The backed-out transaction records are identified and reprocessed.

4. Transaction recovery after an ABORT.

   Performed by TPS. The backed-out transaction records are identified and reprocessed.

5. Journal Rollback

   Performed by TPS. Tank journals can be rolled back at the user's discretion. For the TRHISTORY Journal, TPS recognizes the occurrence of a data base ROLLBACK or the equivalent situation using REBUILD. The TRHISTORY Journal can then be rolled back to remain synchronized with the data base.

6. Disaster Recovery

   Performed by a user-written program. If the data base and its audit files are destroyed or otherwise unusable, an off-line

dump of the data base can be reloaded and all transaction records captured in the TRHISTORY file after a given time can be reprocessed. This process is similar to processing tanked transaction records, which is also a user-controlled process.

7. User-Written Program Restart

With the aid of TPS, the user-written program determines where it was interrupted and continues from that point.

TRANSACTION RECOVERY DESIGN FEATURES

TPS facilitates recovery through the following design features:

1. All data base transactions are designed by the user to be independent of each other. That is, all transactions must be two-phase or must follow an equivalent programmatic convention. A transaction must not use the results of a prior transaction and must keep records locked until END-TRANSACTION. If these conditions are not met, previous results may not be reproduced during TPS recovery. For more details about the requirements for updating a data base using TPS, refer to section 6 of this manual.

2. All input transaction records are saved in a transaction history file named the TRHISTORY journal. Saving this history of transactions guarantees that user-written programs do not have to recreate transactions that have already completed. Furthermore, the order in which the records are saved is the same as the order required to reprocess them in a single-threaded recovery operation that reproduces previous results. The records are ordered by saving each record while the transaction against the data base is passing between the two phases of a two-phase transaction. (For details on two-phase transactions, refer to the discussions of Consistency and Reproducibility in Section 6 of this manual.)

3. DMSII maintains in the data base control file, and writes to the audit file, the TRHISTORY journal address of the transaction record corresponding to each data base transaction. Any recovery operation that moves the data base backward or forward in time maintains this TRHISTORY journal address in the data base control file to synchronize the data base with the TRHISTORY journal.

4. A Halt/Load flag is maintained in the control file of each transaction journal, so that TPS can recognize the occurrence of a Halt/Load.

5. TPS can determine whether or not the user-written program has performed the required sequence of operations, including BEGIN-TRANSACTION, MID-TRANSACTION, and END-TRANSACTION. TPS can also determine whether or not the user-written program is in transaction state. These capabilities allow TPS to recognize the occurrence of an ABORT, or the need to force an ABORT.

6. For each identifiable transaction user, the following information is saved in the journal control file:

a. A reference to either the last successful input transaction record or the last restart transaction record.

b. The response transaction record that corresponds to the last successful input transaction record.

The user can retrieve these records in order to determine where to restart an interrupted program.

TRANSACTION RECOVERY AFTER A HALT/LOAD

A TPS recovery immediately follows the data base RECOVERY. When the Transaction Library is initiated, it checks the status of the Halt/Load flag in the journal control file. This flag is always RESET when the Transaction Library terminates normally. If it is still SET, then the previous use of the Transaction Library was either interrupted by a Halt/Load or DSed.

The Transaction Library must recover the end-of-file pointer for the last allocated journal data file in the currently opened journal to avoid losing the information written before the interruption. Once this end-of-file pointer is recovered, the Transaction Library opens the data base, which causes data base Halt/Load RECOVERY to run if necessary. After RECOVERY is run, the Transaction Library begins to reprocess transactions.

Transaction reprocessing begins by positioning the TRHISTORY journal to a point just beyond the TRHISTORY journal address in the data base control file. Because of the order of transaction records in the TRHISTORY journal, all records preceding this point are in the data base, and all records following this point are not in the data base. The Transaction Library merely reads transaction records and passes them, one at a time, to the user-written Update Library for reprocessing.

The reprocessing operation must be successfully completed to ensure reproduced results. Irrecoverable I/O errors that occur during the RECOVERY operation may prevent results from being reproduced.

Users must wait until the RECOVERY operation is completed to process new transactions.

TRANSACTION RECOVERY AFTER AN ABORT

Transaction recovery after an ABORT is performed immediately after data base ABORT RECOVERY is completed. Transaction recovery after an ABORT is very similar to transaction recovery after a Halt/Load. However, TPS can recognize a Halt/Load more easily than an ABORT. To ensure recognition of an ABORT, the user-written Update Library code must employ the TPS-related forms of BEGIN-TRANSACTION and END-TRANSACTION as well as the new DMSII user language statement MID-TRANSACTION. These statements pass the transaction record to the ACCESSROUTINES, which mark the record to indicate the successful execution of the statements.

Because these statements are executed in the stack in which the Transaction Library is running (although the base SIB is in another stack), the Transaction Library can determine whether the stack is still in transaction state when the user-written Update Library code has completed. If END-TRANSACTION was not executed, and if the stack is no longer in transaction state, then that transaction was ABORTed. If the stack is still in transaction state, then the Transaction Library must force an ABORT by calling another procedure in the Update Library.

Once the Transaction Library recognizes the occurrence of an ABORT or forces an ABORT, other users are locked out and forced to wait until ABORT recovery has completed. After recovery has completed, the user-written program that caused the ABORT is selected to begin reprocessing backed-out transactions, as in Halt/Load recovery. After the ABORT has finished, all users can proceed with their transactions. The user-written program that caused the ABORT then receives an exception from the Transaction Library indicating that the transaction failed in its attempt to update the data base.

JOURNAL ROLLBACK

When the Transaction Library checks for the occurrence of a Halt/Load, it may discover that the data base appears to be at a state prior to the state of the journal. This situation occurs when no Halt/Load has occurred, but the TRHISTORY journal address in the data base control file does not match the address of the last input transaction record saved in the TRHISTORY journal control file.

This mismatch could occur for one of the following reasons:

1. An old data base control file that did not reflect the state of the data base was loaded and initialized, causing the values of the last successful transaction to be lost.

2. The data base has undergone a ROLLBACK or REBUILD.

After the data base has undergone ROLLBACK or REBUILD, the TRUTILITY program (discussed in section 11 of this manual) should be run to accomplish the following:

1. To recover the journal control file so that it is synchronized with the data base.

2. To adjust the end-of-file pointer for the journal data file.

TRUTILITY should be run before allowing any user-written programs to access the transaction base, so that the transaction base and data base do not appear to be out of synchronization.

If the Transaction Library encounters a mismatch between the TRHISTORY journal address in the data base control file and the last transaction record saved in the TRHISTORY journal, the Library notifies the user and asks for instructions to do one of the following:

1. Continue with transaction processing despite the mismatch.

2. DS the user-written program, and allow the user to run the TRUTILITY program to recover the control file before proceeding.

REINITIALIZING AN EXISTING JOURNAL

An existing TRHISTORY journal should be reinitialized using the INITIALIZE statement. (For more information, refer to "TRUtility Statements" in Section 11 of this manual.) Upon reinitialization of the journal, the Transaction Library (when it is first reopened) resets the address of the last normally completed TPS transaction in the data base control file.

If an existing TRHISTORY journal is reinitialized, and a subsequent data base REBUILD or ROLLBACK is performed, then the TRHISTORY journal control file may be at a point in time that bears no relationship to the state of the data base. To ensure synchronization of a reinitialized TRHISTORY journal with the data base, the first step is to perform an off-line dump of all TPS files. Then the following steps should be taken, depending on whether a data base REBUILD or ROLLBACK is going to be performed.

Case 1. REBUILD

A data base REBUILD should be performed in the usual way. Depending upon when the REBUILD completes, one of the following two options should be taken:

1. If the data base REBUILD is completed before the point when the TRHISTORY journal is reinitialized, the dump of TPS files should be loaded and TRUTILITY should be run in order to recover the TRHISTORY journal control file. TPS will then be synchronized with the data base.

2. If the data base REBUILD is completed after the point where the TRHISTORY journal is reinitialized, TRUTILITY should be run (or re-run, if appropriate) to recover the TRHISTORY journal control file. TPS will then be synchronized with the data base.

Case 2. ROLLBACK

A data base ROLLBACK should be performed in the usual way. Depending upon when the data base ROLLBACK completes, one of the following two options should be taken:

1. If the data base ROLLBACK extends to a point in time before the TRHISTORY journal is reinitialized, the dump of TPS files should be loaded and TRUTILITY should be run in order to

recover the TRHISTORY journal control file. TPS will then be synchronized with the data base.

2. If the data base ROLLBACK extends beyond the point when the TRHISTORY journal is reinitialized, TRUTILITY should be run to recover the TRHISTORY journal control file. TPS will then be synchronized with the data base.

RESTARTING OF USER-WRITTEN PROGRAMS

TPS greatly simplifies restart of user-written programs, because a user-written program needs to provide for restart only after a DS or Halt/Load interruption. When restart is necessary, each program must be activated by calling LOGONTRUSER because all users are automatically deactivated following an interruption in processing. TPS saves each user's last successful input transaction record (possibly including the restart transaction record) as well as the corresponding response transaction record. User-written programs can retrieve these transaction records by calling the Transaction Library procedures RETURNRESTARTINFO and RETURNLASTRESPONSE, passing the appropriate user id number. These transaction records contain all the information needed to restart the user-written program.

11 TRUTILITY

Software components provided with TPS include a utility program named TRUTILITY, which the user tailors to a particular transaction base. TRUTILITY allows the user to:

1. Initialize journal control files.
2. Search through and print journal data files for certain transaction records.
3. Print information in a journal control file.
4. Recover a journal control file that has been lost or damaged.

Each of these functions corresponds to a command that the user inputs to TRUTILITY.

COMPILING TRUTILITY

TRUTILITY compilation can be initiated automatically by SYSTEM/TFL whenever a new TRDESCRIPTION file is created. Automatic compilation occurs if the TFL ZIP dollar-card option is set. The title of the resulting TRUTILITY code file is <base name>/CODE/UTILITY.

Alternatively, the user can compile TRUTILITY by compiling the source file TRBASE/UTILITY using the TRBASE/PROPERTIES, DATABASE/DMCONTROL, and TRDESCRIPTION files and the DMALGOL compiler, as shown in Figure 11-1. All required processor options are included in the source file. The internal DMALGOL compiler file named DASDL must be label-equated by the user to the appropriate TRDESCRIPTION file, named <base name>/TRDESCRIPTION. If the TRBASE/PROPERTIES file is not visible to the WFL job compiling the source file, then the compiler file DASDL should be label-equated to the TRBASE/PROPERTIES file. Similarly, if the SYSTEM/TRINTERFACE file is not visible to the WFL job compiling the source file, then the compiler file XCODE must be label-equated to the SYSTEM/TRINTERFACE file.

A compile-time check ensures that the property level of the TRBASE/PROPERTIES file matches the property level of the TRDESCRIPTION file. If the TRBASE/PROPERTIES level exceeds the TRDESCRIPTION file level, the syntax error "DESCRIPTION FILE TOO OLD FOR PROPERTIES" is issued. If the TRDESCRIPTION level exceeds the TRBASE/PROPERTIES level, the syntax error "PROPERTIES TOO OLD FOR DESCRIPTION FILE" is issued.

A Series DMSII Transaction Processing System (TPS)

PCN 1164043-001 Release 3.6 December 1985

Example:

The following example shows a WFL job for compiling the TRUTILITY program.

```
?JOB COMPILEUTILITY;
 USER = SITE;
 FAMILY = TPS OTHERWISE DMS;
 COMPILE BANKTR/CODE/UTILITY WITH DMALGOL [CT]
     LIBRARY;
 OPTION FAULT DSED ARRAYS;
 PL FILE DASDL = BANKTR/TRDESCRIPTION;
 PL FILE TRPROPERTIES = TRBASE/PROPERTIES;
 PL FILE DMCONTROL = DATABASE/DMCONTROL;
 PL FILE TAPE = TRBASE/UTILITY;
 PL FILE ERRORFILE (KIND = DISK, PROTECTION = SAVE,
     TITLE = ERROR/UTIL);
 DATA CARD
 $ MERGE LINEINFO ERRLIST
 IF CT ISNT COMPILEDOK THEN
     ABORT "*** SYNTAX ERROR ***";
?END JOB;
```

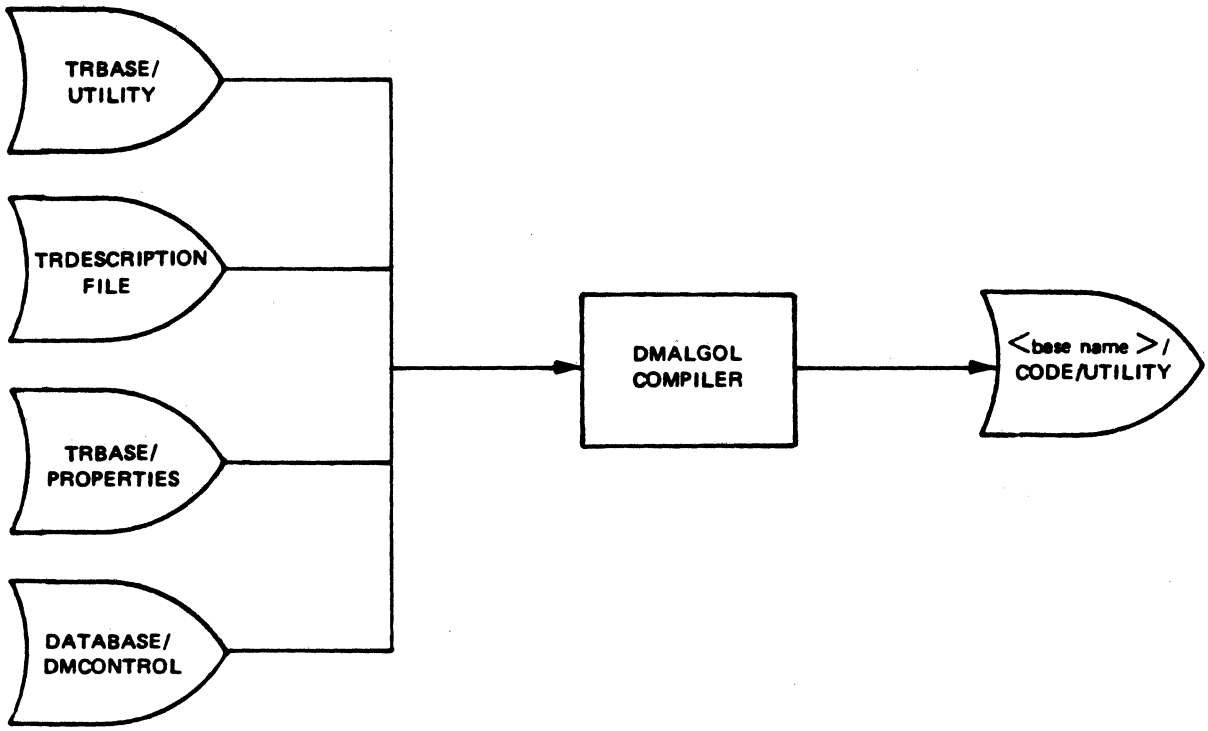


Figure 11-1. Compilation of the TRUTILITY Program

RUNNING TRUTILITY

TRUTILITY can be initiated through CANDE using the following command:

```
RUN <base name>/CODE/UTILITY ("<trutility statement>");
```

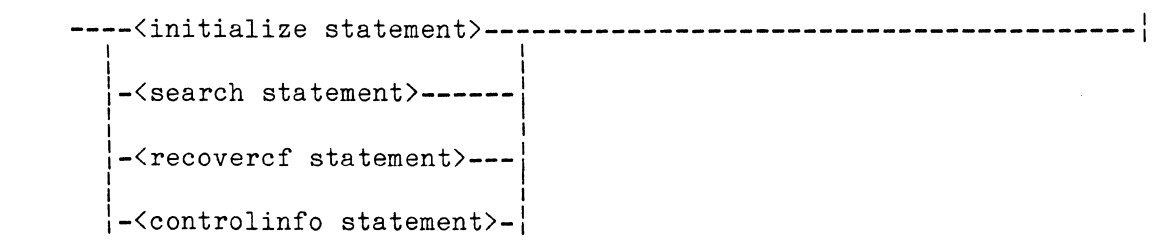
Alternatively, TRUTILITY can be initiated by a WFL job of the following form:

```
<i> RUN BANKTR/CODE/TRUTILITY("*")
<i> DATA
    <trutility statement>
<i> END
```

In either of these forms, <trutility statement> contains the statement that TRUTILITY is to perform.

Syntax:

<trutility statement>



TRUTILITY STATEMENTS

The following list gives a brief description of each trutility statement. Each statement is described in detail later in this section.

<initialize statement>

Initializes a journal control file.

<search statement>

Searches a specified journal data file for certain transaction records and optionally prints them out.

<recovercf statement>

Recovers a specified control file from the journal's data files.

<controlinfo statement>

Prints certain information about a specified journal control file.

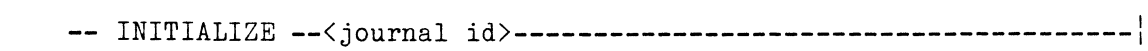
INITIALIZE Statement

The INITIALIZE statement initializes a new journal or reinitializes an existing journal control file. (For more information, refer to "Reinitializing An Existing Journal" in Section 10 of this manual.) TRUTILITY determines whether any attributes were specified explicitly for the journal in the TFL source file. If no attributes have been specified, then TRUTILITY uses the attribute values specified in the TFL defaults statement. If no default attribute values are specified, the standard system default values are used.

TRUTILITY creates a journal control file named <base name>/<journal id>/CONTROL. Journal data files are subsequently created by the Transaction Library as needed. The user passes <journal id> in a user-written program as the LIBPARAMETER attribute to the Transaction Library. For more information concerning LIBPARAMETER, refer to Section 6 of this manual.

A transaction journal must be INITIALIZEd before it can be used. If an existing journal is reinitialized, then the old control file is removed and all existing data files are made inaccessible to the Transaction Library.

Syntax:



<journal id>

A <simple identifier> that identifies the name of the transaction journal to be initialized. (For more information on <simple identifier>s, refer to "TFL Basic Constructs" in Section 3 of this manual.)

SEARCH Statement

The SEARCH statement causes TRUTILITY to search a specified transaction data file for certain transaction records and optionally print them out. Functionally, the print mechanism is similar to that used by the DMSII PRINTAUDIT utility. However, TRUTILITY can print transaction records in specific formats, as the DMSII INQUIRY program does.

For any SEARCH request, the following information must be given:

1. The journal to be searched.
2. A range to be searched.
3. The users whose records are examined.
4. A criterion for specifying which transaction records are selected.

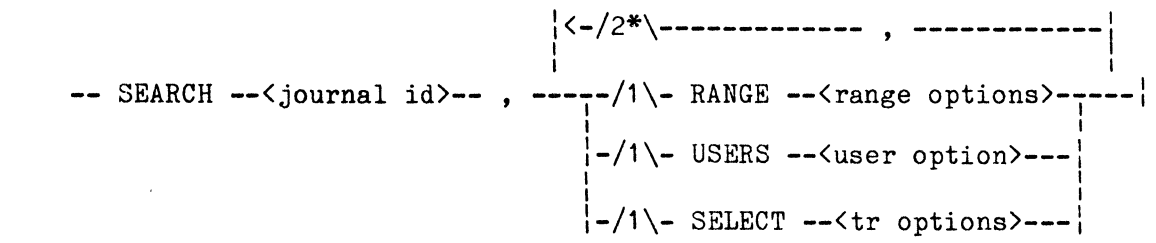
The transaction record formats or subformats, or both, can be selected in one of the following ways:

1. All transaction record formats and subformats that are encountered can be selected.
2. A list of the desired transaction record formats and subformats can be provided to select only members of the list.

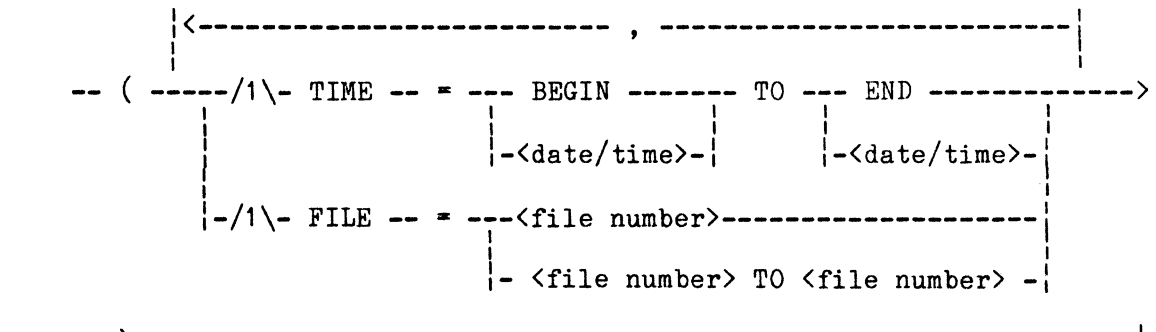
   The selection can be further limited by specifying a condition involving items in transaction records in addition to the transaction record format and optional subformat.

Selected records can be printed, counted or both. Records can be printed in hexadecimal format or in a formatted layout in which all of the items in the selected formats and subformats are printed along with their values.

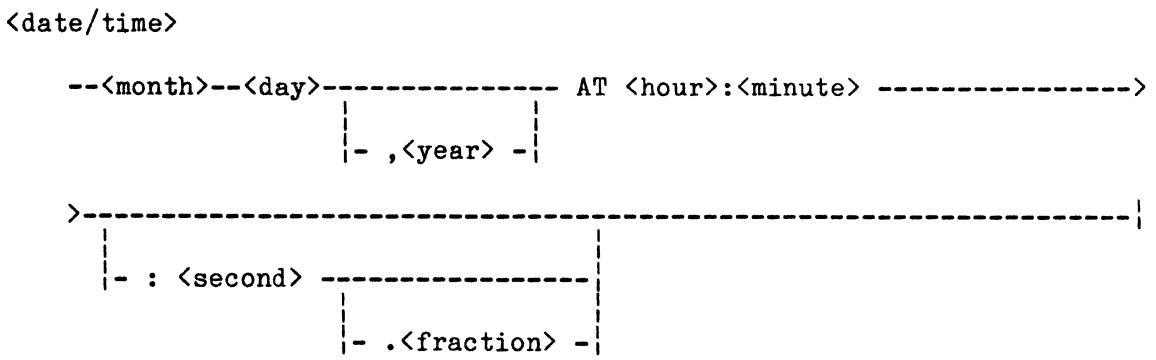
Syntax:



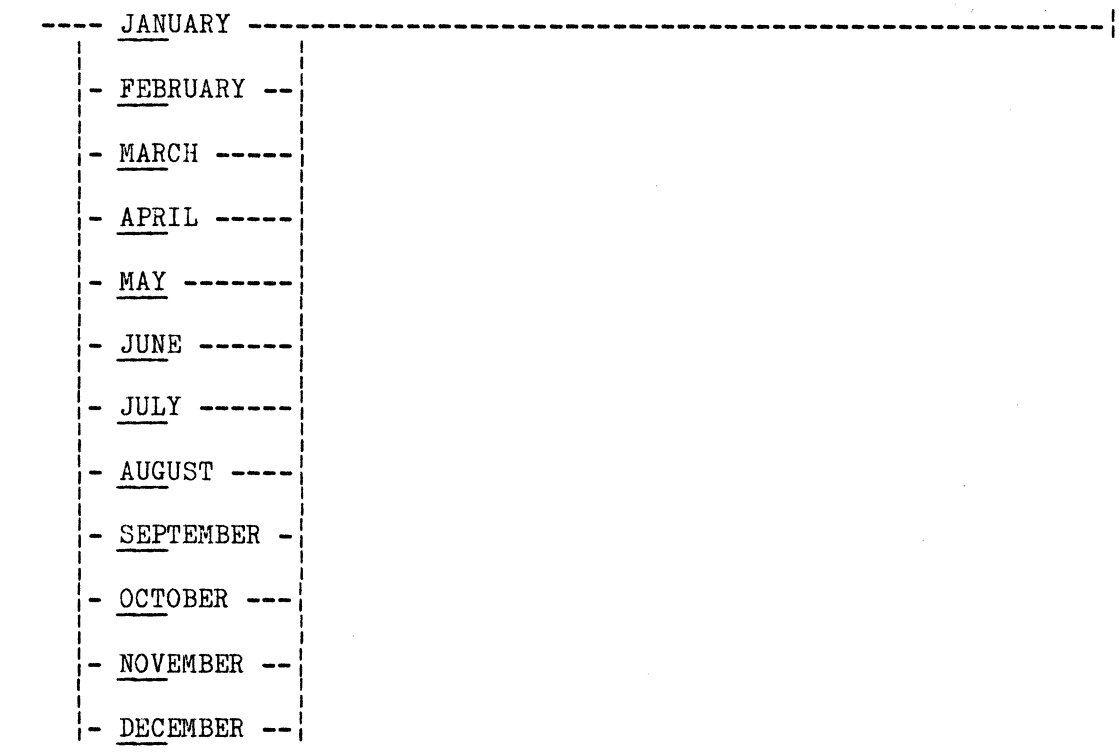
<range options>



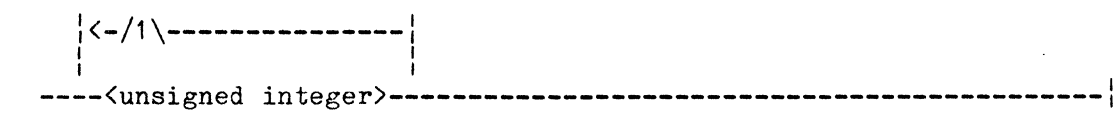
<date/time>



<month>



<day>



<year>

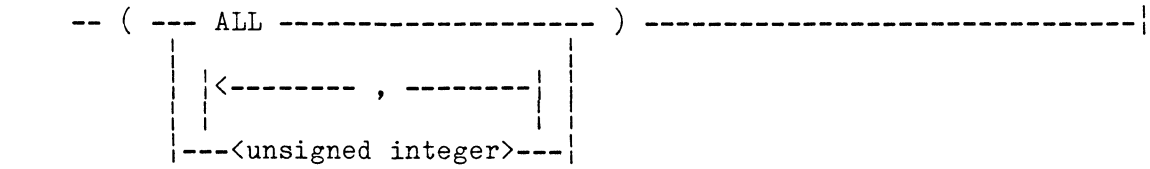
<hour>

<minute>

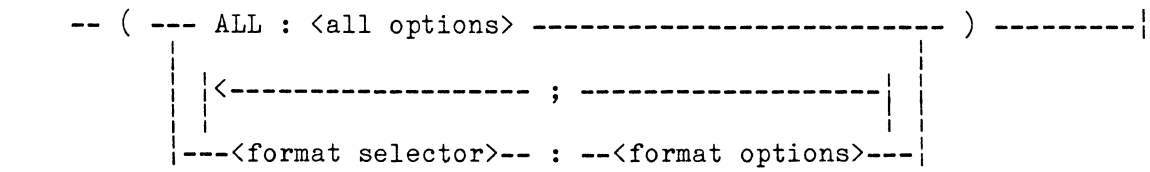
<second>

<fraction>

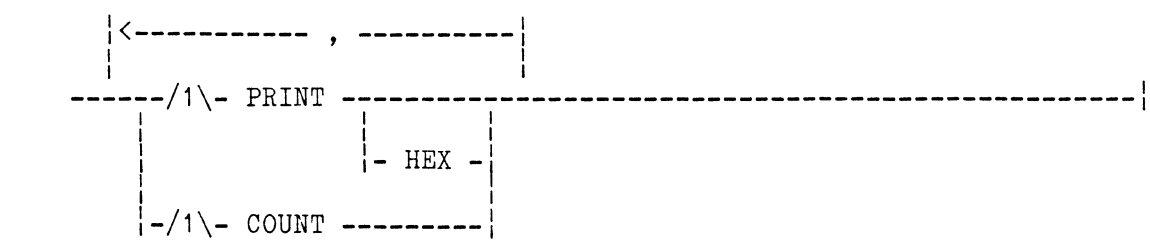
<user option>



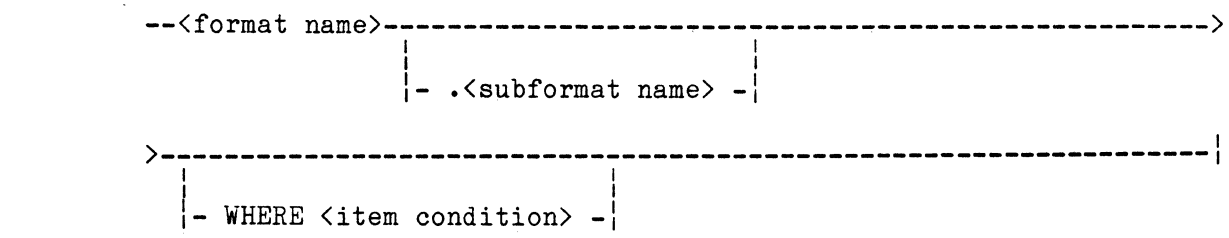
<tr options>



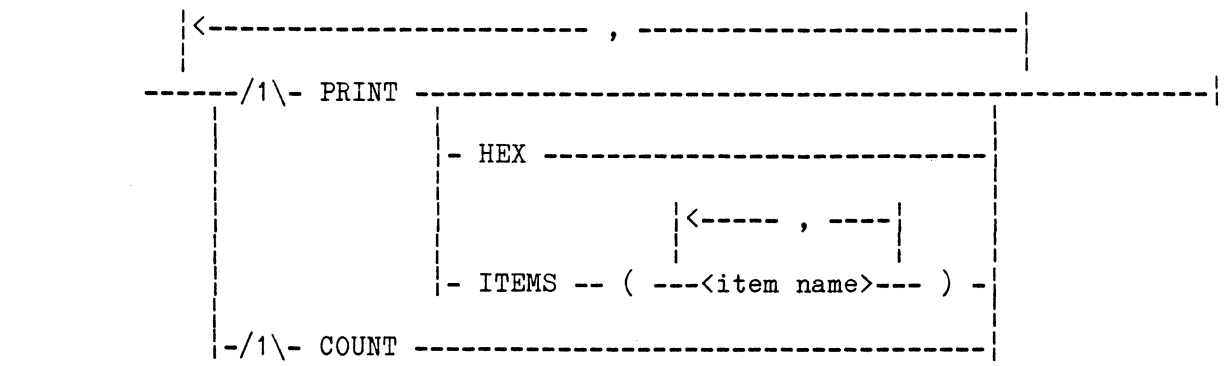
<all options>



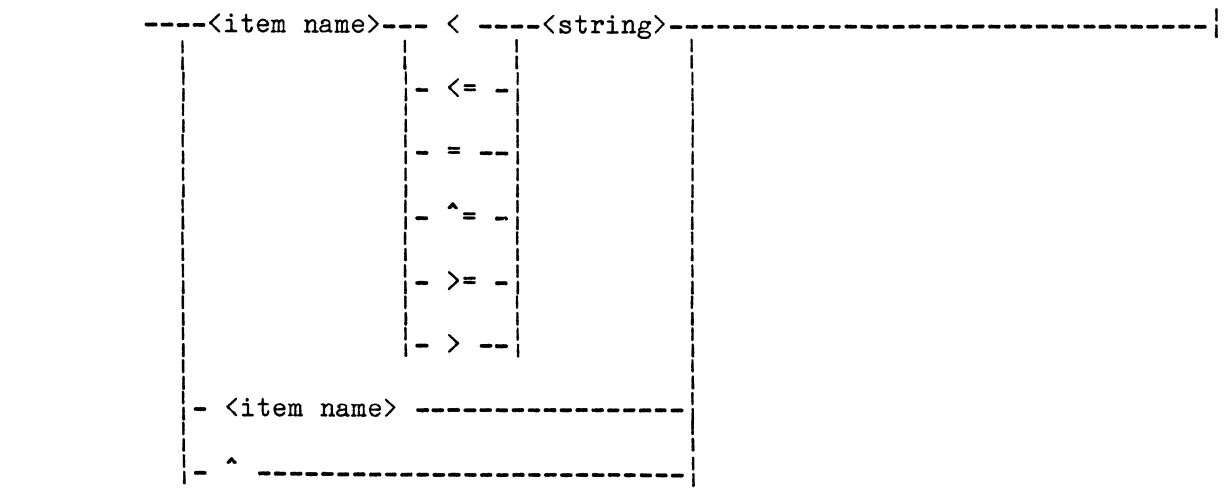
<format selector>



<format options>



<item condition>



Semantics:

<journal id>

A <simple identifier> that identifies the transaction journal to be searched. For more information about <simple identifier>s, refer to "TFL Basic Constructs" in Section 3 of this manual.

RANGE <range options>

Causes TRUTILITY to search for records in the specified time range or file range, or both.

USERS <user option>

Causes TRUTILITY to search for records within the specified time or file range that correspond to the specified range of user id numbers. These user id numbers are the id numbers that are returned from the CREATETRUSER entry point in the Transaction Library.

SELECT <tr options>

Causes TRUTILITY to select records within the search range that correspond to the specified transaction formats and, optionally, subformats.

(TIME = BEGIN TO END)
(TIME = BEGIN TO <date/time>)
(TIME = <date/time> TO END)
(TIME = <date/time> to <date/time>)

Causes TRUTILITY to search for records in the time range delimited by the specified options: the beginning of the journal (BEGIN), the end of the journal (END), or a particular <date/time> within the journal.

(FILE = <file number>)
(FILE = <file number> TO <file number>)

Causes TRUTILITY to search for records in the journal data file specified by <file number> or in the range of journal data files between the first and second <file number>s.

(ALL)
(<unsigned integer>, <unsigned integer> ...)

Causes TRUTILITY to search for records corresponding to given user id numbers. If ALL is specified, the range includes all user id numbers known to the specified transaction journal. If a list of user ids is specified, the search only selects records corresponding to those user id numbers.

(ALL:<all options>)
(<format selector>:<format options>; ...)

Causes TRUTILITY to select specific transaction formats and, optionally, subformats. If ALL:<all options> is specified, all formats and subformats are selected. If a list of formats and optional subformats is specified, only they are selected.

PRINT
PRINT HEX
PRINT ITEMS (<item name>)
COUNT

Specifies what is to be done with the selected formats and subformats. If PRINT is specified, the selected formats and subformats are printed in a formatted layout. If the HEX option is also included, the selected formats and subformats are printed in hexadecimal. If the ITEMS <item name> option is specified, only those items listed are printed. <item name> identifies a name in the transaction record format or subformat. (More than one <item name> may be specified.) If the COUNT option is specified, the selected formats and subformats are counted, and a summary of all counts is output to the line printer.

<format name>
<format name>.<subformat name>

Identifies the particular formats and, optionally, subformats that are selected.

WHERE <item condition>

Selects only the formats and subformats for which the <item condition> gives a value of TRUE.

In the <item condition>, <item name> identifies an item in the transaction record format or subformat.

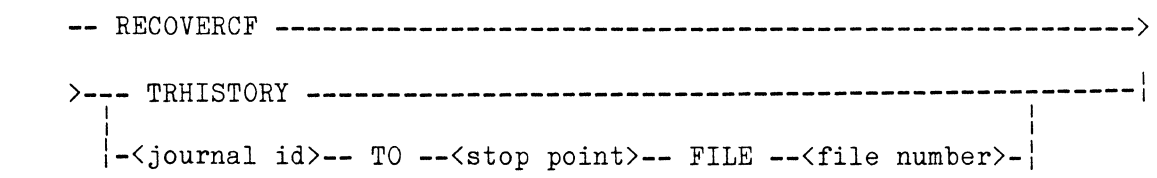
For <item name> or <item name> in the lower part of the syntax diagram, the <item name> must be either a Boolean or a bit in a field of bit items.

RECOVERCF Statement

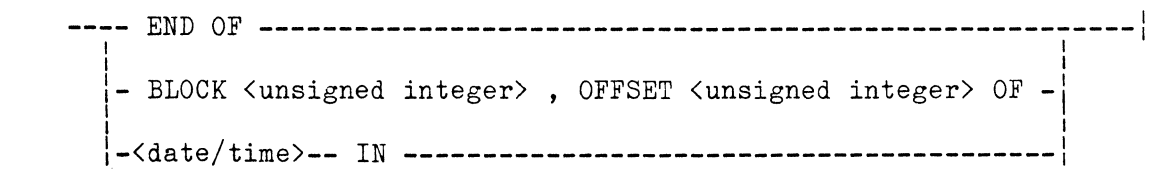
The RECOVERCF statement recovers a journal control file from the journal data files if the existing control file is missing or corrupted. This statement can also be used to help roll a transaction journal back to an earlier point in time.

Syntax:

<recovercf statement>



<stop point>



Semantics:

RECOVERCF TRHISTORY

Rolls back the TRHISTORY journal to a point where it is synchronized with the data base.

When the Transaction Library is invoked, it does not run if it is not synchronized with the data base. This failure to run could be caused by a ROLLBACK of the data base. Under these circumstances, RECOVERCF TRHISTORY should be run.

RECOVERCF TRHISTORY can also be used when the TRHISTORY journal control file has been lost or destroyed. First the user should recover a journal control file to work with, preferably a copy of a recent one that the user dumped. If no copy is available, the user could initialize the journal control file to create one. Executing the RECOVERCF TRHISTORY function then synchronizes the journal control file with the data base, using the TRHISTORY journal data files to reflect changes to the journal control file up to the current point in time.

RECOVERCF <journal id> TO <stop point> FILE <file number>

Recovers the control file for a tank journal. Because tank files are not synchronized with any part of TPS, the user must provide a <stop point> to which the control file is recovered.

<journal id>

A <simple identifier> that identifies the transaction journal that is recovered. (For a definition of <simple identifier>, refer to "TFL Basic Constructs" in Section 3 of this manual.)

<file number>

An integer value identifying a particular journal data file. Recovery occurs up to the <stop point> specified within this file.

<stop point>

Identifies the point in the journal data file to which the journal control file is recovered. This can be a specific date and time, a block and offset within the the journal data file, or the end of the journal data file. TRUTILITY uses the journal data files to update the journal control file so that the journal control files reflect the state of the tank journal at the specified <stop point>.

By specifying an earlier <date/time>, the user can roll back a tank file to a particular point in time. In this way, a series of transactions that are not to be processed can be discarded. This function cannot be used for the TRHISTORY journal.

END OF

Specifies that the end of the specified journal data file is used as the <stop point> in recovering a journal control file.

BLOCK <unsigned integer>

Specifies the beginning segment address of a block within a journal data file.

OFFSET <unsigned integer>

Specifies a word offset within the block specified by BLOCK <unsigned integer> that is used as the <stop point> in recovering a journal control file.

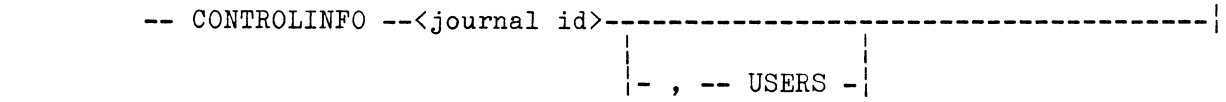
<date/time>

Specifies a date and time to be used as the <stop point> in recovering a journal control file. A month, day and year can be specified along with a time (which includes the hour and minute, and optionally, the second and fraction of a second). For the syntax of <date/time> refer to "SEARCH Statement" in this section.

CONTROLINFO Statement

The CONTROLINFO statement prints information about a journal control file.

Syntax:



Semantics:

CONTROLINFO <journal id>
CONTROLINFO <journal id>, USERS

Prints information in the control file of a journal. The table of contents of the first block of the control file is printed by default. The USERS option causes information to be printed about each known user for the journal, including the last response transaction record and the location of the last restart (or possibly input) transaction record.

<journal id>

A <simple identifier> that identifies the journal whose control file information is printed. (For more information about <simple identifier>s, refer to "TFL Basic Constructs" in section 3 of this manual.)

12. CONVERTING TO TPS

This section discusses the conversion of DMSII user language programs for an existing data base to the use of TPS. Conversion can take one of the following two forms:

1. Converting all such programs to the use of TPS.

2. Converting some of these programs to the use of TPS, but continuing to run others as conventional DMSII user language programs. (Running in this mode is referred to in this section as "mixed mode.")

When the number of user-written programs for an existing DMSII data base is small, all update programs should be converted to use TPS. In this case, TPS guarantees reproducible results after reprocessing transactions, provided that the user has programmed the Update Library for two-phase transactions. (For more information on reproducibility and two-phase transactions, refer to Section 6 of this manual.)

When a large number of user-written programs access an existing data base, converting to mixed mode may be more desirable. However, TPS cannot guarantee reproducible results when mixed mode is used unless certain restrictions are imposed. (For more information on restrictions, refer to Restrictions That Ensure Reproducibility in this section.) Because of this difficulty with reproducibility, converting to mixed mode should be considered a temporary solution, unless reproducibility is not an important user requirement.

CONVERTING TO MIXED MODE

TPS makes the conversion to mixed mode easier by allowing programs that access a data base through TPS and conventional DMSII programs to update a data base simultaneously. To permit simultaneous access, the DASDL global data base option TPSDUALUPDATE must be set. When TPSDUALUPDATE is set, a data base can be opened by both the conventional DMSII statement OPEN UPDATE and the TPS statement OPEN TRUPDATE.

The TPSDUALUPDATE option is reset by default. If this option is reset, the data base can be opened by either conventional DMSII statements or TPS statements, but not by both. An attempt to use both types of OPEN statement results in an OPENERROR user language exception. During the conversion process (while mixed mode is still in effect) the TPSDUALUPDATE option should be SET. After all DMSII programs have been converted to TPS, the option should be RESET. (For more information about the TPSDUALUPDATE option, refer to the DMSII DASDL Reference Manual).

MIXED MODE RESTRICTIONS THAT ENSURE REPRODUCIBILITY

If all user-written programs access the data base through TPS and the Update Library has been programmed for two-phase transactions, then TPS guarantees reproducible results. But if other non-TPS programs are also running against the data base, even if they have been programmed for independence, TPS cannot guarantee reproducibility unless certain restrictions are imposed. These restrictions are required because the results from a TPS transaction that requires the same data base records as a non-TPS program running at the same time may not be reproduced if the transactions have to be resubmitted following a data base ABORT or a Halt/Load. The Transaction Library can resubmit only the transactions it has saved in the TRHISTORY journal, and the original sequence of events may not be reproduced.

The following programming conventions can help to achieve reproducibility while mixed mode is in effect:

1. Create two sets of transactions that are independent of each other. This is the simplest restriction that can be imposed. The user-written programs should be divided into two groups that access disjoint portions of the data base. This division allows one set of programs to be run through TPS and the other set of programs to be run as conventional DMSII programs without affecting reproducibility of the TPS programs, because the order in which the non-TPS transactions are reprocessed is unrelated to the TPS transactions.

2. Divide the user-written programs into two groups that have the least amount of overlap, and then take additional steps to ensure reproducibility. To solve the problem of programs that access common portions of the data base, particular operational guidelines can be followed to ensure reproducibility. Running the programs that would destroy independence only when TPS is not running is an example of an operational strategy.

3. If imposing operational guidelines is not feasible, user-written programs can be coded to ensure that if a TPS program is referencing a certain portion of the data base, then a conventional DMSII update program cannot reference that portion of the data base at the same time. A data set with a single record could be used as a lock to accomplish this.

   In addition, the programming convention followed must ensure that a syncpoint is caused when the program is finished running. In this way, a subsequent program cannot cause its transactions to be backed out.

   This programming convention is considered the least desirable of the three alternatives discussed above. Other programming conventions that achieve a degree of independence comparable to those suggested in the preceding paragraphs might also be feasible.

A HOW TO READ RAILROAD DIAGRAMS

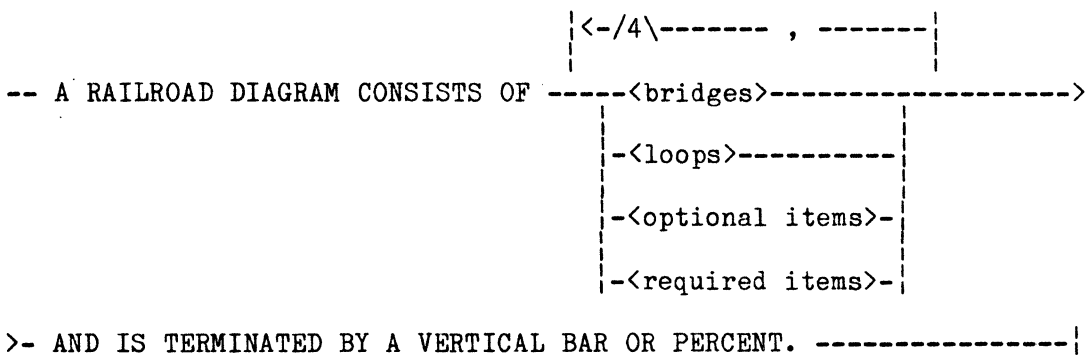
Railroad diagrams graphically represent the syntax of software commands.

The railroad diagrams are traversed left to right or in the direction of the arrowhead. Adherence to the limits illustrated by bridges produces a syntactically valid statement. Continuation from one line of a diagram to another is represented by a right arrow (>) appearing at the end of the current line and the beginning of the next line. The complete syntax diagram is terminated by a vertical bar (!) or a percent sign (%).

Items contained in broken brackets (<>) are syntactic variables which are defined in the manual or are information which the user is required to supply.

Uppercase items not enclosed in broken brackets must appear literally; the minimum abbreviations are underlined.

Example:



The following are some of the syntactically valid constructs that may be generated from the preceding diagram:

A RAILROAD DIAGRAM CONSISTS OF <bridges> AND IS TERMINATED BY A VERTICAL BAR OR PERCENT.

A RAILROAD DIAGRAM CONSISTS OF <optional items> AND IS TERMINATED BY A VERTICAL BAR OR PERCENT.

A RAILROAD DIAGRAM CONSISTS OF <bridges>, <loops> AND IS TERMINATED BY A VERTICAL BAR OR PERCENT.

A Series DMSII Transaction Processing System (TPS)

PCN 1164043-001 Release 3.6 December 1985

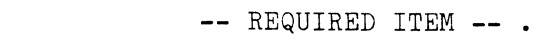
A RAILROAD DIAGRAM CONSISTS OF <optional items>, <required items>, <optional items>, <bridges>, <loops> AND IS TERMINATED BY A VERTICAL BAR OR PERCENT.

Railroad Components

<required items>

No alternate path through the railroad diagram exists for required items or required punctuation.

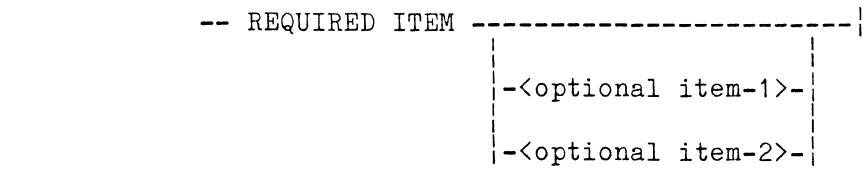
Example:



<optional items>

Items shown as a vertical list indicate that the user may make a choice of the items specified. An empty path through the list allows the optional item to be absent.

Example:



The following valid constructs may be generated from the preceding diagram:

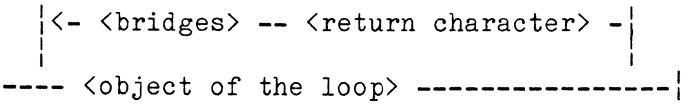
REQUIRED ITEM

REQUIRED ITEM <optional item-1>

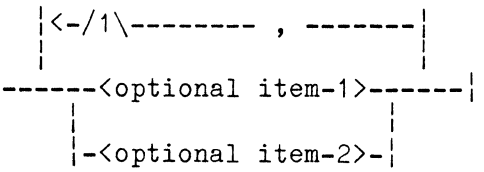
REQUIRED ITEM <optional item-2>

<loops>

A loop is a recurrent path through a railroad diagram and has the following general format:



Example:



The following are some of the valid constructs that may be generated from the preceding diagram:

<optional item-1>

<optional item-1>,<optional item-1>

<optional item-2>,<optional item-1>

The loop must be traversed in the direction in which the arrowhead points, and the limits specified by bridges cannot be exceeded.

<bridges>

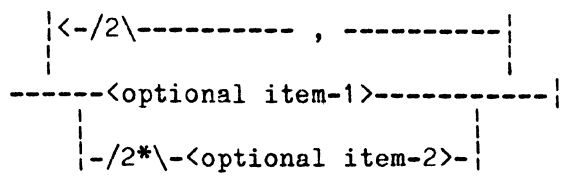
A bridge illustrates the minimum or maximum number of times a path may be traversed in a railroad diagram.

Two forms of bridges exist:

/n\ n is an integer that specifies the maximum number of times the path may be traversed.

/n*\ n is an integer that specifies the maximum number of times the path may be traversed. The asterisk (*) indicates that the path must be traversed at least once.

Example:



The loop may be traversed a maximum of two times, and the path for <optional item-2> must be traversed at least once but no more than twice.

The following are some of the valid constructs that may be generated from the preceding diagram:

<optional item-1>,<optional item-2>

<optional item-2>,<optional item-2>,<optional item-1>

<optional item-2>

B GLOSSARY

This glossary contains definitions for many of the terms used in this manual. The terms are defined in alphabetical order. Each term that has an associated acronym is spelled out; the acronym follows the term in parentheses. (For example: "Message Control System (MCS).") Many terms have, in addition to their definitions, a list of related definitions. These related definitions are indicated by a "see also" list following the main definition.

ACCESSROUTINES — Routines for a data base description that perform all physical and logical management of the data base and allow many users to access the data base concurrently. Each data management statement executed by a user language program invokes a portion of the ACCESSROUTINES to perform all of the file management functions required by the statement.

address — The identification of the position of a location in storage.

alpha item — A data item that stores alphanumeric information as EBCDIC characters. = (See also: Extended Binary Coded Decimal Interchange Code (EBCDIC))

audit trail — A file produced by the ACCESSROUTINES that contains various control records and a sequence of "before" and "after" record images resulting from changes to the data base. The audit trail is used to recover the data base and supply restart information to programs after a hardware or software failure has occurred.

(See also: ACCESSROUTINES, data base)

back out — A recovery operation in which the changes made against the data base are undone and the progress of one or more transactions is rolled back to a consistent state at an earlier point in time.

(See also: rollback)

A Series DMSII Transaction Processing System (TPS)

bit

The fundamental unit of computer information, which is a single binary digit. A bit can be in one of two conditions: OFF (corresponding to a value of binary 0) or ON (corresponding to a value of binary 1).

block

A group of physically adjacent records that are packaged together so that they can be transferred to or from a physical file as a group.

(See also: physical file, record)

BLOCKSIZE attribute

A file attribute that gives the length of a file block.

(See also: block, file attribute)

BNA

Burroughs Network Architecture

Boolean item

A data item that stores information having a logical value of either TRUE or FALSE.

(See also: data item)

byte

A measurable portion of consecutive bits. On Burroughs Large Systems, a byte is eight bits in length.

(See also: bit)

CANDE

Command and Edit Language. A Burroughs MCS that provides generalized file preparation and updating capabilities, in an interactive, terminal-oriented environment.

(See also: Message Control System (MCS))

checksum

A value used to detect certain classes of I/O errors. A checksum is computed for each data base file block by applying an equivalence operator to each word in the block. When the block is physically written, the checksum value is stored in a

checksum word appended to the end of the block. When the block is read, the checksum is recomputed and the result is compared to the stored value. A checksum error occurs if the two values are not equal.

(See also: data base, word)

compiler

A program that converts computer instructions written in a source language into machine code.

control file

A file containing data file coordination information, audit control information, and dynamic data base parameter values. The control file also provides data base interlock control; that is, it allows functions such as RECOVERY to have exclusive use of the data base. In addition, the control file provides program-to-file and file-to-file compatibility.

(See also: transaction journal, journal)

control item

A system-defined item contained in a transaction record. A control item is maintained by the Transaction Processing System and is read-only in all BDMSCOBOL, BDMSALGOL, and BDMSCOBOL74 programs. The initial value of a control item is assigned when a transaction record is created. Some types of control items are of little interest to the user program; other types must be used in order to process the transaction records properly. A control item is defined only after a transaction record has been CREATEd with a particular format and possibly subformat.

For details on control items, refer to Section 4 of this manual.

control word

The first word of a control item used for integrity checking.

(See also: control item)

coroutine

A process in which two programs execute in parallel, but not simultaneously, pass control back and forth to each other, and run alternately.

D1 (D[1]) stack

A stack that is created for each code file being executed, which contains one entry for each segment of object code that was created when the program was compiled. Each entry in the segment dictionary indicates whether or not the associated segment is in memory and contains the disk or memory address at which the segment begins. The terms "segment dictionary" and "D1 (D[1]) stack" are synonyms.

In the TPS Transaction Library, a D1 (D[1]) and a D2 (D[2]) stack are shared by all users of a transaction base.

D2 (D(2)] stack

A stack that is initiated for each executing program. The working stack is used for storage of non-array variables and of descriptors for array variables. When a computation requires preliminary calculation of intermediate results, these results are placed on top of the working stack. In addition, the working stack is used to handle all procedure calls in the program. The terms "working stack" and "D2 (D[2]) stack" are synonyms.

For TPS, data and procedures that are global to a particular transaction base reside in the Transaction Library's D2 (D[2]) stack. The D2 (D[2]) and D1 (D[1]) stacks are shared by all users of a transaction base. The D2 (D[2]) stack is also known as the <transaction base name>/CODE/HOSTLIB stack.

D3 (D[3]) stack

The Transaction Library stack that is initiated for each active transaction journal for a particular transaction base. Data and procedures that are local to a particular transaction journal reside in the D3 (D[3]) stack. The D3 (D[3]) stacks are also known as the <transaction journal name>/SHARED/LIBRARY stacks. Each D3 (D[3]) stack is shared by all users of that

journal.

D4 (D[4]) stack

=The Transaction Library stack where entry points reside and are called by user-written programs. Entry points in the D4 (D[4]) stack can call procedures in the D2 (D[2]) and D3 (D[3]) stacks. Data and procedures that are local to a single Transaction Library invocation reside in the D4 (D[4]) stack. This stack is also known as the <transaction journal name>/PRIVATE/LIBRARY stack.

Data and Structure Definition Language (DASDL)

=The Burroughs language used to describe a Large Systems data base. DASDL is the source language that is input to the DASDL compiler, which derives the data base description file from the input.

(See also: data base, description file)

data item

A field in a data base record or transaction format that contains a particular type of information.

(See also: data base, record)

data set

A collection of related data records stored in a file on a random-access storage device. A data set is similar to a conventional file. It contains data items and has logical and physical properties similar to files. However, unlike conventional files, data sets may contain other data sets, sets, and subsets.

data base

A centralized collection of data placed into one or more files. More than one user-written program can access this data at the same time; thus, redundant files are not required for each application. User-written programs running in batch, time-sharing, and remote job entry environments may all access a data base concurrently.

description file — The file produced by the DASDL or TFL compiler that contains information used when compiling all tailored software and all DMSII user-language programs for a particular data base or transaction base.

(See also: Data and Structure Definition Language (DASDL), data base, transaction base, Transaction Formatting Language (TFL))

disjoint — The adjective used to describe a data set, set, or subset that is not contained in another data set.

(See also: data set, set, subset)

DMSII — Data Management System II

Extended Binary Coded Decimal Interchange Code (EBCDIC) — An 8-bit code representing 256 graphic characters.

embedded — The adjective used to describe a data set, set, or subset that is contained in another data set.

(See also: data set , set , subset)

entry point — A procedure that allows access to a separately compiled library code file.

EOF — End-of-file.

field item — A data item that can contain unsigned integer and Boolean values.

(See also: data item)

file — A group of related records.

(See also: record)

file attribute — A system control parameter that contains

	the information needed to connect a file on the system (a "physical file") to a file declared in a program (a "logical file"). This information can include the name of the file, the peripheral device associated with the file, or the size of records in the file. (Refer to the I/O Subsystem Reference Manual for a complete description of file attributes and their uses.)
global data item	A data item, group item, or population item that is not a part of any data set. Global data items generally contain information such as control totals, hash totals, and populations applying to the entire data base. (See also: data item, data base, group item, population)
group item	A data item that contains a collection of subordinate data items and group items. (See also: data item)
guardfile	A file created by the utility program SYSTEM/GUARDFILE, which describes the access rights of various users and programs to a program or data file or a data base.
Halt/Load	A system-initialization procedure that loads a fresh version of the MCP from disk or pack to main memory. (See also: Master Control Program (MCP))
host system	In a Shared Resources or BNA environment, the system (processor) that controls the data base. (See also: remote system)

initial value

In TPS, the value assigned to an item in a newly-created transaction record. An initial value can be declared for each item in the TFL declaration for the item; otherwise, TFL assigns the item a default initial value.

internal file name

The name used to declare a file in a program. The internal name of a file is given by the value of its INTNAME file attribute.

(See also: file attribute, logical file.)

I/O

Input/Output.

journal control file

A control file containing global information about the journal it resides in, as well as information about users known to the journal, and response transaction records for each known user.

journal data file

A file within a transaction journal that contains a serial history of transaction records.

key item

A data item or group item that serves as a key of retrieval for a set, subset, or access.

library

A separately compiled code file that user programs can access by calling procedures known as entry points that have been defined within the code file.

(See also: entry point.)

logical data base

A collection of structures declared in the DASDL that can be used to enforce structure-level security and achieve data independence.

When each logical data base is declared in DASDL, the data sets, sets, subsets, and

	remaps that are to be included in it are listed. Programs can only use the structures contained in the logical data base that they invoke. Thus, the data base administrator may control which structures each program accesses by limiting the use of logical databases and the physical data base through guardfiles.
logical file	A file variable declared within a program, which represents the file and its structure to the program. A logical file has no properties of its own until it is described by file attributes or associated with a physical file; a physical file inherits properties from the logical file that creates it. More than one logical file can be associated with a single physical file. (See also: file attribute, physical file)
master	The adjective used to describe a data set or record that contains one or more embedded data sets or records. "Master," "parent," and "owner" are synonyms.
MCP	Master Control Program
member	A record of a data set.
Message Control System (MCS)	A special-purpose DCALGOL program that controls the flow of messages between terminals and the main system. Burroughs message control systems include SYSTEM/CANDE, SYSTEM/RJE, SYSTEM/GEMCOS, and SYSTEM/DIAGNOSTICMCS.
multiprogramming	A technique for handling multiple routines or programs simultaneously by overlapping or interleaving their execution (in other words, permitting more than one program to time-share machine components).
null	The value contained in a data base item when it does not contain valid information.

numeric item	A data item that stores signed or unsigned packed decimal values.
owner	Refer to "master."
path	A specific location within the logical ordering of a data set, set, subset, or access.
physical file	A file as it is stored on a particular recording medium such as disk or tape, or the physical device on which a file is stored. (See also: file, logical file)
RDS	Restart Data Set
real item	Data items that store signed or unsigned, fractional or whole values in single-precision, floating-point form. (See also: data item)
record	A collection of data items that are usually related in some logical way. (See also: data item)
record-level locking	The exclusive use of a data base record by a process until that record is explicitly or implicitly freed.
RECOVERY	A data base routine that is initiated after a hardware, software, or operations failure while the data base is in update mode. RECOVERY backs out any partially completed transactions by applying audit-trail images to the data base to restore it to its proper state. In addition, RECOVERY passes restart information to the programs accessing the data base.
remap	A logical data record that redefines a

	physical data set record by omitting, reordering, or renaming the items. (See also: data set, record)
Remote Library	A TPS module that enables transaction processing from remote systems in a data communications network. The Remote Library is compiled on the host system, then copied over to each remote system. Thus, a program on a remote system calls the same entry points that a program on the host system calls, except that the program on a remote system is actually calling the Remote Library.
remote system	In a Shared Resources or BNA environment, all systems (processors) connected to the system that controls the data base. (See also: host system)
rerun mode	The state during which TPS resubmits transactions that have been backed out of the data base after a data base ABORT or Halt/Load.
restart	To return to a particular point in a program and resume operation from that point.
restart data set	A data set that stores restart records that users can access to recover data base information in the event of a system failure.
restart record	A record in a transaction journal containing user-defined information that allows a user program to restart in response to a particular condition. A restart record is saved in the transaction journal along with the corresponding input transaction record.

rollback	Recovery of a data base or transaction base to a consistent state at an earlier point in time.
serial processing	Time-sequential reading or writing of a record following the one currently being processed.
set	A file of indices that refer to all of the records of a single data set. Sets are automatically maintained by the system. Sets permit access to the records of a data set in some logical sequence and are normally used to optimize certain types of retrievals of the data set records.
Shared Resources	A distributed processing environment, in which users share all resources throughout a network, including processors, peripherals, terminals, databases, and information.
SIB	Structure Information Block
source program	A program coded in a language other than machine language that must be translated into machine language before use.
stack	A special array of words occupying some location in memory. Entries in a stack are always addressed as displacements from the fixed location known as "bottom-of-stack." (See also: word)
STATISTICS	A parameter in the TFL source that causes the Transaction Library to maintain information concerning the run-time performance of a transaction base. This information is printed when the Transaction Library is closed.

structure

A data set, set, subset, access, or remap.

(See also: data set, remap, set, subset)

subformat

(See: transaction subformat)

subrecord description

A list of item descriptions, optionally followed by a VERIFY condition. For any transaction record, the applicable VERIFY condition(s) must yield the value TRUE when evaluated using the items of that record. Any given record may have one or two VERIFY conditions, or no VERIFY condition at all. If any VERIFY condition yields the value FALSE, the record is not acceptable.

(See also: transaction record, VERIFY clause)

subset

A structure that is identical to a set except that it need not contain a record for every record of the data set, and it may or may not be automatically maintained by DMSII.

(See also: data set, set)

symbolic file

A file that contains a source program.

(See also: source program)

syncpoint

A point in time when no program updating the data base is in transaction state and a syncpoint record is written to the audit trail. Syncpoints are enforced by the ACCESSROUTINES and occur at intervals specified in DASDL or forced programmatically using an ENDTRANSACTION statement with the SYNC option. When a syncpoint is pending, no program may enter transaction state until all programs that are already in transaction state have left it, a syncpoint record has been written to the audit trail, and the audit buffers have been flushed. When a Halt/Load occurs, RECOVERY eliminates any partially completed

transactions by backing up the data base to the last point at which no program was in transaction state. Syncpoints limit the number of transactions which will be backed out of the data base. In particular, if SYNCPOINT*11N, no program has more than N transactions backed out by ABORT or Halt/Load RECOVERY.

(See also: ACCESSROUTINES, audit trail, Halt/Load, syncpoint, transaction state)

tanking

In TPS, the operation in which the Transaction Library stores transactions in a tank journal and does not process them against the data base. A tank journal can be any transaction journal except the TRHISTORY journal.

(See also: journal, transaction journal)

TPS

Transaction Processing System

TFL

Transaction Formatting Language

transaction

A unit of processing against a DMSII data base that constitutes a single logical change to the data base. A TPS transaction is performed by a transaction processing routine. It may include, but is not to be confused with, a data base transaction, which is user-written code delimited by BEGIN-TRANSACTION and END-TRANSACTION.

transaction base

The software and files that constitute a TPS interface to a data base. A transaction base can contain a maximum of 4095 transaction record formats.

(See also: transaction record format)

transaction format

(See: transaction record format)

Transaction Formatting Language (TFL)
: The Burroughs language used to write source files that are compiled to produce description files for transaction bases.

  (See also: description file, transaction base)

transaction journal
: A collection of one control file and any number of data files, used to store information about transactions.

  (See also: journal control file, journal data file)

Transaction Library
: The Transaction Library is a collection of procedures that are accessed by user-written programs to process or tank transactions and to read them back from transaction journals. The procedures are accessed through a set of entry points supplied by the Transaction Library, which is tailored for particular transaction base during compilation.

  (See also: Update Library)

transaction processing routine
: A routine that can retrieve data from a DMSII data base or update a DMSII data base. A transaction processing routine takes a single transaction record as input and produces a single transaction record as output.

transaction record
: A structured variable that contains user-defined data items and system-defined control items for individual transactions. The user-defined items are similar to the data items in a DMSII data set record or a COBOL 01-level variable. A transaction record can be passed as a parameter to and from a procedure. It can also be read from and written to a transaction journal.

transaction record format
: A TFL construct that defines the format of

	a single transaction record, including the data items and group items that can be contained within it.
transaction state	The period in a user-language program between a BEGINTRANSACTION statement and an ENDTRANSACTION statement. Programs making changes to an audited data base are required to be in transaction state when the changes are made.
transaction subbase	A TFL construct for a transaction base that includes a subset of the formats and possibly subformats declared for the transaction base. (See also: Transaction Formatting Language (TFL), transaction base)
transaction subformat	The variable part of a transaction record format.
two-phase transaction	A transaction in which the first execution phase locks records without freeing any, the second and final execution phase of the transaction frees records without locking any, and no records are retrieved without locking them.
update	The process of modifying the information in a data base or transaction base.
Update Library	A collection of user-written processing routines that serve as an interface between the Transaction Library and a DMSII data base. These processing routines can be written in any one of the DMSII user languages: ALGOL, COBOL74 or COBOL. The Update Library is the only user-written module within TPS that contains the data base declaration and all of the code that performs data management statements against the data base. (See also: Transaction Library)

usercode	An entry in the system USERDATAFILE identifying the user as a valid user and identifying files in the user's library. (See also: USERDATAFILE)
user transaction	A single input transaction record.
variable-format data set	A data set whose records consist of two parts, a fixed part and a variable-format part. A single record description exists for the fixed part. The variable-format part can describe several variable parts. An individual record is constructed by using the fixed part alone, or by joining the fixed part with one of the variable parts. (See also: record)
VERIFY clause	A clause containing conditions (known as VERIFY conditions) which is declared in the TFL source to ensure that only valid transaction records are accepted by the Transaction Library.
WFL	Work Flow Language
word	A unit of computer memory. On Burroughs Large Systems, a word consists of 48 bits that are used for storage plus 3 tag bits, used to indicate how the word is interpreted, and 1 parity bit, used for parity checking.
work area	A memory area in a program where data records are constructed, accessed, or modified.

APPENDIX C. EXAMPLE OF A TRANSACTION BASE

The following is an example of a typical DMSII application using the Transaction Processing System. The example centers around a simple banking application in which bank accounts are created and deleted, deposits and withdrawals are made, and account balances are determined. An example is given for each user-supplied piece of software: a DASDL description, a TFL description, a user-written program and an Update Library. The user-written program and the Update Library are written in both ALGOL and COBOL.

In addition to the banking example, an ALGOL procedure is given which illustrates "detanking." This procedure reads transactions from a tank journal and processes them against the data base.

DASDL FOR BANKING EXAMPLE:

```
OPTIONS(AUDIT);
PARAMETERS(SYNCPOINT = 10 TRANSACTIONS);

ACCOUNT DATA SET              % SPECIFY A DATA SET TO HOLD THE ACCOUNT
(                             % NUMBERS AND INFO. ASSOCIATED WITH THEM.
    ACCOUNT-NUM NUMBER(6);
    NAME ALPHA(20);
    BALANCE REAL(S10,2);

    DEPOSIT UNORDERED DATA SET  % USED TO KEEP HISTORY OF THE DEPOSITS
    (                           % AND WITHDRAWALS MADE.
        TRANDATE REAL;
        OLD-BALANCE REAL(S10,2);
        AMOUNT REAL(S10,2);     % NEGATIVE FOR WITHDRAWAL
        NEW-BALANCE REAL(S10,2);
    );

);

ACCOUNT-SET SET OF ACCOUNT
    KEY ACCOUNT-NUM;

RDS RESTART DATA SET          % REMEMBER: MUST SPECIFY A RESTART DATASET
(
    X ALPHA(10);
);
```

TFL DESCRIPTION OF TRANSACTION BASE

```
BANKTR TRANSACTION BASE;     % FIRST DECLARE THE NAME OF THE TRANSACTION
                             % BASE WE ARE ABOUT TO DESCRIBE.
PARAMETERS
(
    STATISTICS,
    DATABASE = BANKDB ON DISK,
    RESTARTDATASET = RDS,
    HOSTSYSTEM = SYS456
);

DEFAULTS          % HERE WE SPECIFY DEFAULTS FOR ITEMS OF TRANSACTION
                  % FORMATS AND FOR JOURNAL CONTROL AND DATA FILES.
(
    ALPHA (INITIALVALUE = BLANKS),
    BOOLEAN (INITIALVALUE = FALSE),
    NUMBER (INITIALVALUE = 0),
    REAL (INITIALVALUE = 0),
    CONTROL FILE
    (
        AREAS = 100,
        AREASIZE = 100 BLOCKS,
        BLOCKSIZE = 20 SEGMENTS,
        FAMILY = DISK,
        CHECKSUM = TRUE
    ),
    DATA FILE
    (
        AREAS = 100,
        AREASIZE = 100 BLOCKS,
        BLOCKSIZE = 30 SEGMENTS,
        FAMILY = DISK,
        CHECKSUM = TRUE
    )
);

CREATEACCT TRANSACTION FORMAT      % THE FOLLOWING FORMATS ARE
(                                  % USED IN THE APPLICATION
    ACCTNUM NUMBER(6);             % PROGRAM AND THE UPDATE
    NAME ALPHA(20);                % LIBRARY.
);

PURGEACCT TRANSACTION FORMAT
(
    ACCTNUM NUMBER(6);
);

DEPOSIT TRANSACTION FORMAT
```

Example of a Transaction Base

```
(
    ACCTNUM NUMBER(6);
    TRANDATE REAL;
    AMOUNT REAL(10,2);
);

WITHDRAWAL TRANSACTION FORMAT
(
    ACCTNUM NUMBER(6);
    AMOUNT REAL(10,2);
    TRANDATE REAL;
);


STATUS TRANSACTION FORMAT
(
    ACCTNUM NUMBER(6);
    BALANCE REAL(S10,2);
    G GROUP
    (  A ALPHA(20);
       B REAL; );
);

RESTARTDETANKER TRANSACTION FORMAT   % THIS FORMAT ILLUSTRATES POSSIBLE
(                                    % INFORMATION TO BE KEPT IN A
    TANKFILENUM FIELD(14);           % RESTART TRANSACTION RECORD.
    TANKBLOCKNUM FIELD(32);
    TANKOFFSET FIELD(16);
);

MANAGER TRANSACTION SUBBASE     % EXAMPLE SUBBASE THAT A MANAGER MIGHT
(                               % USE. NOTE THAT A GUARDFILE IS ATTACHED
    CREATEACCT,                 % TO IT FOR SECURITY.
    PURGEACCT,
    DEPOSIT,
    WITHDRAWAL,
    STATUS,
),
    GUARDFILE = BANKTR/MANAGER/GUARDFILE;

TELLER TRANSACTION SUBBASE     % EXAMPLE SUBBASE THAT A TELLER MIGHT USE
(
    DEPOSIT,
    WITHDRAWAL,
    STATUS
);


TRHISTORY TRANSACTION JOURNAL  % THIS IS AN EXAMPLE OF SPECIFYING
CONTROL FILE                   % EXPLICIT VALUES FOR THE ATTRIBUTES OF
(                              % THE TRHISTORY JOURNAL.
    AREAS = 100,
    AREASIZE = 100 BLOCKS,
```

```
    BLOCKSIZE = 20 SEGMENTS,
    FAMILY = DISK,
    CHECKSUM = TRUE
),
DATA FILE
(
    AREAS = 100,
    AREASIZE = 2 BLOCKS,
    BLOCKSIZE = 3 SEGMENTS,
    FAMILY = DISK,
    CHECKSUM = TRUE
);


TANK1 TRANSACTION JOURNAL     % EXAMPLE OF TANK JOURNAL ATTRIBUTE
CONTROL FILE                  % SPECIFICATION.
(
    USERCODE = SAMPLEUSER,
    FAMILY = PACK
),
DATA FILE
(
    USERCODE = SAMPLEUSER,
    DUPLICATED ON DISK
);
```

EXAMPLES OF USER-WRITTEN PROGRAMS

The following programs written in COBOL and ALGOL show how TPS can be used for a number of simple banking transactions. For the COBOL programs, card columns 1 through 6 are not shown; all COBOL code represented here ostensibly begins in card column 7.

COBOL

```
$ SET FREE
 IDENTIFICATION DIVISION.
 ENVIRONMENT DIVISION.
 INPUT-OUTPUT SECTION.
 FILE-CONTROL.
     SELECT REMOTEFILE
        ASSIGN TO REMOTE.
     SELECT PRINTFILE
        ASSIGN TO PRINTER.
 I-O-CONTROL.
 DATA DIVISION.
 FILE SECTION.
 FD  REMOTEFILE
     RECORD CONTAINS 72 CHARACTERS.
 01  RMT-REC                                PIC X(72).
 01  RMT-FIELDS.
     03 RMT-1                            PIC X(10).
     03 RMT-2                            PIC X(10).
     03 RMT-3                            PIC X(10).
     03 RMT-4                            PIC X(10).
     03 RMT-5                            PIC X(10).
     03 RMT-6                            PIC X(10).
     03 RMT-7                            PIC X(12).
 FD PRINTFILE
     RECORD CONTAINS 132 CHARACTERS.
 01  PRINT-REC                             PIC X(132).

*
**********************************************************************
*
*   DECLARE THE TRANSACTION BASE TO BE USED AND ASSIGN AN INTERNAL
*   NAME TO THE "STATUS" FORMAT.
*
**********************************************************************
*
 TRANSACTION SECTION.
 TB  TRB = BANKTR
        USING STATIS=STATUS, CREATEACCT, RESTARTDETANKER,
              DEPOSIT, WITHDRAWAL, PURGEACCT, SYSTEMTR.
 01  TRIN.
```

```
 01  TROUT.
 01  LASTINPUT.
 01  RESTARTTRREC.
 01  LASTRESPONSE.
*
 WORKING-STORAGE SECTION.
 77  USER-N PIC 999 VALUE 0.
 77  USER-ID   PIC X(7).
 01  JOBPARAMCOMP COMP WITH LOWER-BOUNDS.
     03  FILLUR PIC 9(11) OCCURS 20.
 01  JOBPARAM REDEFINES JOBPARAMCOMP.
     03  JOBPARAMCHAR PIC X OCCURS 99.
 77  IDNUM         COMP      PIC 9(8).
 77  N             COMP      PIC 9(8).
 77  X             COMP      PIC 9(8).
 77  OPT           COMP      PIC 9(8).
 77  TANKING       COMP      PIC 9(8).
 77  ACCT                    PIC X(6).
 77  TPS-RESULT    COMP      PIC 9(8).
 77  TOTAL         COMP      PIC 9(6).
 77  TIMEOUTV      COMP      PIC 9(8).
 77  USER-NAME               PIC X(20).
 77  OPT-X                   PIC X(1).
 77  AMT                     PIC X(6).
 77  FILENUMBER    COMP      PIC 9(4).
 77  BLOCKNUMBER   COMP      PIC 9(8).
 77  OFFSETS       COMP      PIC 9(8).
 77  TEMP                    PIC X(4).
 77  ID-X                    PIC X(17).
 77  STATOPTION              PIC 9(4).
 01  SP.
     03  SPCHAR PIC X OCCURS 80.
 01  LIBPARAM PIC LX(100) DEPENDING ON N.
***
**************************************************************************
*
 PROCEDURE DIVISION.
 MAIN SECTION.
 P1.
     OPEN I-O REMOTEFILE.
     OPEN OUTPUT PRINTFILE.
     MOVE 3 TO TANKING.

*
**************************************************************************
*
*  SET THE LIBPARAMETER BEFORE THE FIRST CALL ON A LIBRARY ENTRY
*  POINT.
*
**************************************************************************
*
     MOVE "JOURNAL NAME?" TO RMT-REC.
     PERFORM WRITE-RMT.
```

```
     READ REMOTEFILE INTO USER-NAME ; INVALID GO TO ERR.
     CHANGE ATTRIBUTE LIBPARAMETER OF "BANKTR/CODE/HOSTLIB"
        TO USER-NAME.
*
     MOVE "WHAT DO YOU WANT TO DO?" TO RMT-REC.
     PERFORM WRITE-RMT.
     MOVE  "CHOICE (1=UPDATE, 2=INQUIRY, 3=TANK, 4=READ)"
        TO RMT-REC.
     PERFORM WRITE-RMT.
     READ REMOTEFILE INTO OPT-X  ; INVALID KEY GO TO EXT.
     MOVE OPT-X TO OPT.
     MOVE OPT TO RMT-REC. PERFORM WRITE-RMT.
     PERFORM OPENTRBASE.
     IF TPS-RESULT > 0 GO TO ERR.
*
**********************************************************************
*
*  CREATE A RESTART TRANSACTION TO BE WRITTEN TO THE TRHISTORY
*  FILE ALONG WITH THE INPUT TRANSACTION. NOTE THAT WE DO NOT
*  ASSIGN VALUES TO THE ITEMS OF THIS RECORD. NORMALLY ONE WOULD
*  DO SO, BUT FOR SIMPLICITY WE LEFT THAT CODE OUT OF THIS
*  EXAMPLE.
*
**********************************************************************
*
     CREATE RESTARTDETANKER IN RESTARTTRREC.
     MOVE "USER ID ? (X17)" TO RMT-REC.
     PERFORM WRITE-RMT.
     READ REMOTEFILE INTO ID-X  ; INVALID KEY GO TO EXT.
     PERFORM LOGONTRUSER.
     IF TPS-RESULT  > 0 GO TO ERR.
     MOVE IDNUM TO RMT-2.
     PERFORM WRITE-RMT.

*
 LOOP.
     MOVE "FUNCTION NAME (OR HELP)" TO RMT-REC.
     PERFORM WRITE-RMT.
     READ REMOTEFILE INTO SP  ; INVALID KEY GO TO EXT.
     IF SP = "CREATEUSER"   PERFORM  CREATEUSERP              ELSE
     IF SP = "PURGEUSER"    PERFORM  PURGEUSERP               ELSE
     IF SP = "CREATE"       PERFORM  CREATEP                  ELSE
     IF SP = "PURGE"        PERFORM  PURGEP                   ELSE
     IF SP = "DEPOSIT"      PERFORM  DEPOSITP                 ELSE
     IF SP = "WITHDRAWAL"   PERFORM  WITHDRAWLP               ELSE
     IF SP = "QUIT"         PERFORM  QUITP                    ELSE
     IF SP = "STAT"         PERFORM  STATUSP                  ELSE
     IF SP = "NEWUSER"      PERFORM  NEWUSERP                 ELSE
     IF SP = "REOPEN"       PERFORM  REOPENP                  ELSE
     IF SP = "SEEK"         PERFORM  SEEKP                    ELSE
     IF SP = "R"            PERFORM  READP                    ELSE
     IF SP = "HELP"         PERFORM  HELPP                    ELSE
     IF SP = "GETLAST"      PERFORM  GETLASTP THRU GETLASTP-EXIT
```

```
                                                                ELSE
     IF SP = "SW"           PERFORM  SWITCHP                    ELSE
     IF SP = "STAT"         PERFORM  STATISTICSP                ELSE
     DISPLAY "DID NOT RECOGNIZE FUNCTION NAME".
     GO TO LOOP.
 EXT.
     STOP RUN.

*
**************************************************************************
*
*
*
*
 CREATEP.
*********
     MOVE "FUNCTION IS CREATE" TO RMT-REC.
     PERFORM WRITE-RMT.
     PERFORM GETACCT.
     MOVE "CUSTOMER NAME? (X17) " TO RMT-REC.
     PERFORM WRITE-RMT.
     READ REMOTEFILE INTO USER-NAME  ; INVALID KEY GO TO EXT .
     CREATE CREATEACCT IN TRIN.
     MOVE ACCT TO ACCTNUM OF CREATEACCT OF TRIN.
     MOVE USER-NAME TO NAME OF CREATEACCT OF TRIN.
     PERFORM PROCESSTR THRU PROCESSTR-EXIT.
*
*
 PURGEP.
********
     MOVE "FUNCTION IS PURGE" TO RMT-REC.
     PERFORM WRITE-RMT.
     PERFORM GETACCT.
     CREATE PURGEACCT IN TRIN.
     MOVE ACCT TO ACCTNUM OF PURGEACCT OF TRIN.
     PERFORM PROCESSTR THRU PROCESSTR-EXIT.
*
*
 STATUSP.
*********
     MOVE "FUNCTION IS STATUS" TO RMT-REC.
     PERFORM WRITE-RMT.
     PERFORM GETACCT.
     CREATE STATIS IN TRIN.
     MOVE ACCT TO ACCTNUM OF STATIS OF TRIN.
     PERFORM PROCESSTR THRU PROCESSTR-EXIT.
*
*  IF WE'RE NOT TANKING THEN DISPLAY STATUS OF AN ACCOUNT.
*
     IF OPT NOT EQUAL TANKING
       PERFORM DISPLAYSTATUS.
*
*
```

Example of a Transaction Base

```
 DEPOSITP.
***********
     MOVE "FUNCTION IS DEPOSIT" TO RMT-REC.
     PERFORM WRITE-RMT.
     PERFORM GETACCT.
     MOVE "AMOUNT OF DEPOSIT? (NNNNNN) " TO RMT-REC.
     PERFORM WRITE-RMT.
     READ REMOTEFILE INTO AMT  ; INVALID KEY GO TO EXT .
     CREATE DEPOSIT IN TRIN.
     MOVE ACCT TO ACCTNUM OF DEPOSIT OF TRIN.
     MOVE TIME(1) TO TRANDATE OF DEPOSIT OF TRIN.
     MOVE AMT TO   AMOUNT OF DEPOSIT OF TRIN.
     PERFORM PROCESSTR THRU PROCESSTR-EXIT.
*
*  IF WE'RE NOT TANKING THEN DISPLAY THE STATUS OF AN ACCOUNT.
*
     IF OPT NOT EQUAL TANKING
       PERFORM DISPLAYSTATUS.

*
*
 WITHDRAWLP.
************
     MOVE "FUNCTION IS WITHDRAWAL" TO RMT-REC.
     PERFORM WRITE-RMT.
     PERFORM GETACCT.
     MOVE "AMOUNT OF WITHDRAWAL? (NNNNNN) " TO RMT-REC.
     PERFORM WRITE-RMT.
     READ REMOTEFILE INTO AMT  ; INVALID KEY GO TO EXT .
     CREATE WITHDRAWAL IN TRIN.
     MOVE ACCT TO ACCTNUM OF WITHDRAWAL OF TRIN.
     MOVE  AMT TO AMOUNT  OF WITHDRAWAL OF TRIN.
     MOVE TIME(1)  TO TRANDATE OF WITHDRAWAL OF TRIN.
     PERFORM PROCESSTR THRU PROCESSTR-EXIT.
*
*  IF WE'RE NOT TANKING THEN DISPLAY STATUS OF AN ACCOUNT.
*
     IF OPT NOT EQUAL TANKING
        PERFORM DISPLAYSTATUS.
*
*
 NEWUSERP.
***********
     MOVE "FUNCTION IS NEW USER" TO RMT-REC.
     PERFORM WRITE-RMT.
     MOVE "USER ID ? (X17)  " TO RMT-REC.
     PERFORM WRITE-RMT.
     READ REMOTEFILE INTO ID-X  ; INVALID KEY GO TO EXT.
     PERFORM LOGONTRUSER.
     IF TPS-RESULT > 0 GO TO ERR.
     MOVE "USER# =" TO RMT-1.
     MOVE IDNUM TO RMT-2.
     PERFORM WRITE-RMT.
```

```
*
*
 REOPENP.
*********
      MOVE "FUNCTION IS REOPEN" TO RMT-REC.
      PERFORM WRITE-RMT.
      PERFORM CLOSETRBASE.
      IF TPS-RESULT > 0 GO TO ERR.
      MOVE "WHAT DO YOU WANT TO DO?" TO RMT-REC.
      PERFORM WRITE-RMT.
      MOVE "CHOICE (1=UPDATE,2=INQ,3=TANK,4=READ)" TO RMT-REC.
      PERFORM WRITE-RMT.
      READ REMOTEFILE INTO OPT-X  ; INVALID KEY GO TO EXT .
      MOVE OPT-X TO OPT.
      PERFORM  OPENTRBASE.
      IF TPS-RESULT > 0 GO TO ERR.

*
*
 SEEKP.
********
      MOVE "FUNCTION IS SEEK" TO RMT-REC.
      PERFORM WRITE-RMT.
      MOVE "ENTER FILENUM (NNNN) " TO RMT-REC.
      PERFORM WRITE-RMT.
      READ REMOTEFILE INTO TEMP        ; INVALID KEY GO TO EXT .
      MOVE TEMP TO FILENUMBER.
      MOVE "ENTER BLOCK (NNNN) " TO RMT-REC.
      PERFORM WRITE-RMT.
      READ REMOTEFILE INTO TEMP        ; INVALID KEY GO TO EXT .
      MOVE TEMP TO BLOCKNUMBER.
      MOVE "ENTER OFFSET (NNNN) " TO RMT-REC.
      PERFORM WRITE-RMT.
      READ REMOTEFILE INTO TEMP     ; INVALID KEY GO TO EXT .
      MOVE TEMP TO OFFSETS.
      PERFORM SEEKTRANSACTION.
      IF TPS-RESULT > 0 GO TO ERR.
*
*
 READP.
********
      MOVE "FUNCTION IS READ" TO RMT-REC.
      PERFORM WRITE-RMT.
      PERFORM READTRANSACTION.
      IF TPS-RESULT > 0 GO TO ERR.
      MOVE "F/B/O =" TO RMT-1.
      MOVE TRFILENUM(TRIN) TO RMT-2.
      MOVE TRBLOCKNUM(TRIN)  TO RMT-3.
      MOVE TROFFSET(TRIN)  TO RMT-4.
      MOVE TRFORMAT(TRIN)  TO RMT-5.
      MOVE TRSUBFORMAT(TRIN)  TO RMT-6.
      PERFORM WRITE-RMT.
*
```

```
*
 CREATEUSERP.
**************
     MOVE "FUNCTION IS CREATEUSER" TO RMT-REC.
     PERFORM WRITE-RMT.
     MOVE "USER ID ? (X17) " TO RMT-REC.
     PERFORM WRITE-RMT.
     READ REMOTEFILE INTO ID-X  ; INVALID KEY GO TO EXT.
     PERFORM CREATETRUSER.
     IF TPS-RESULT > 0 GO TO ERR.
     PERFORM LOGONTRUSER.
     IF TPS-RESULT > 0 GO TO ERR.
     MOVE "USER# =" TO RMT-1.
     MOVE IDNUM TO RMT-2.
     PERFORM WRITE-RMT.
*
*
 PURGEUSERP.
*************
     MOVE "FUNCTION IS PURGE USER" TO RMT-REC.
     PERFORM WRITE-RMT.
     PERFORM PURGETRUSER.
     IF TPS-RESULT > 0   GO TO ERR.

*
*
 QUITP.
********
     MOVE "FUNCTION IS QUIT" TO RMT-REC.
     PERFORM WRITE-RMT.
     PERFORM CLOSETRBASE.
     GO TO EXT.
*
*
 SWITCHP.
**********
     MOVE "FUNCTION IS SWITCH" TO RMT-REC.
     PERFORM WRITE-RMT.
     PERFORM SWITCHTRFILE.
     IF TPS-RESULT > 0 GO TO ERR.
*
*
 STATISTICSP.
**************
     MOVE "FUNCTION IS STATISTICS" TO RMT-REC.
     PERFORM WRITE-RMT.
     MOVE "WHAT DO YOU WANT TO DO?" TO RMT-REC.
     PERFORM WRITE-RMT.
     MOVE "CHOICE(1=PRINT & RESET, 2=PRINT, 3=RESET)"
        TO RMT-REC.
     PERFORM WRITE-RMT.
     READ REMOTEFILE INTO TEMP; INVALID KEY GO TO
        EXT.
```

```
      MOVE TEMP TO STATOPTION.
      PERFORM HANDLESTATISTICS.
      IF TPS RESULT > 0 GO TO ERR.
*
*
 HELPP.
********
      MOVE "FUNCTIONS : TEST,CREATE,PURGE,DEPOSIT" TO RMT-REC.
      PERFORM WRITE-RMT.
      MOVE "            WITHDRAWAL,QUIT,STATUS,NEWUSER" TO RMT-REC.
      PERFORM WRITE-RMT.
      MOVE "            REOPEN,SEEK,READ,GETLAST" TO RMT-REC.
      PERFORM WRITE-RMT.
      MOVE "            CREATEUSER,PURGEUSER,SWITCH" TO RMT-REC.
      PERFORM WRITE-RMT.
      MOVE "STATISTICS,HELP."                         TO RMT-REC.
      PERFORM WRITE-RMT.
*
*
 ERR.
*****
      MOVE "RESULT =" TO RMT-1.
      MOVE TPS-RESULT TO RMT-2.
      PERFORM WRITE-RMT.
      GO TO LOOP.
*
*
 WRITE-RMT.
**********
      WRITE RMT-REC.
      WRITE PRINT-REC FROM RMT-REC.
      MOVE SPACES TO RMT-REC.
*
*
*
 PROCESSTR.
**********
      IF OPT NOT EQUAL TANKING GO TO PROCESSTRANS.
         PERFORM TANKTRNORESTART.
         IF TPS-RESULT > 0 GO TO ERR.
         GO TO PROCESSTR-EXIT.
 PROCESSTRANS.
         PERFORM PROCESSTRANSACTION.
         IF TPS-RESULT > 0 GO TO ERR.

 PROCESSTR-EXIT.
*
*
 GETACCT.
*********
      MOVE "ACCOUNT NUMBER ? (NNNNNN) " TO RMT-REC.
      PERFORM WRITE-RMT.
      READ REMOTEFILE INTO ACCT  ; INVALID KEY GO TO EXT.
```

```
*
*
 GETLASTP.
***********
     IF OPT NOT EQUAL TANKING   GO TO GETLASTRESTRTREC.
        PERFORM RETURNRESTARTINFO.
        IF TPS-RESULT > 0 GO TO ERR.
        MOVE "LAST RESTART (FILE,BLK,OFFSET,FMT):" TO RMT-REC.
        PERFORM WRITE-RMT.
        MOVE TRFILENUM(TROUT) TO RMT-2.
        MOVE TRBLOCKNUM(TROUT)  TO RMT-3.
        MOVE TROFFSET(TROUT)  TO RMT-9.
        MOVE TRFORMAT(TROUT)  TO RMT-5.
        PERFORM WRITE-RMT.
        GO TO GETLASTP-EXIT.
 GETLASTRESTRTREC.
        PERFORM RETURNRESTARTINFO.
        IF TPS-RESULT > 0 GO TO ERR.
        MOVE "LAST RESTART (FILE,BLK,OFFSET,FMT):" TO RMT-REC.
        PERFORM WRITE-RMT.
        MOVE TRFILENUM(TROUT) TO RMT-2.
        MOVE TRBLOCKNUM(TROUT)  TO RMT-3.
        MOVE TROFFSET(TROUT)  TO RMT-4.
        MOVE TRFORMAT(TROUT)  TO RMT-5.
        PERFORM WRITE-RMT.
*
        PERFORM RETURNLASTRESPONSE.
        MOVE "LAST RESPONSE(FILE,BLK,OFFSET,FMT)" TO RMT-REC.
        PERFORM WRITE-RMT.
        MOVE TRFILENUM(TROUT) TO RMT-2.
        MOVE TRBLOCKNUM(TROUT)  TO RMT-3.
        MOVE TROFFSET(TROUT)  TO RMT-4.
        MOVE TRFORMAT(TROUT)  TO RMT-5.
        PERFORM WRITE-RMT.
 GETLASTP-EXIT.
*
*
 DISPLAYSTATUS.
***************
     IF TRFORMAT(TROUT) NOT EQUAL TRFORMAT(STATIS) GO TO ERR.
     MOVE "ACCOUNT # " TO RMT-1.
     MOVE ACCTNUM OF STATIS OF TROUT TO RMT-2.
     MOVE "CUR BAL =" TO RMT-3.
     MOVE BALANCE OF STATIS OF TROUT TO TOTAL.
     MOVE TOTAL TO RMT-4.
     PERFORM WRITE-RMT.

*
*
*
*****************************************************************
*
*    THE FOLLOWING ARE THE LIBRARY ENTRY POINT CALLS
```

```
*
*****************************************************************
*
*
 CREATETRUSER.
     CALL "CREATETRUSER        OF BANKTR/CODE/HOSTLIB"
        USING ID-X , IDNUM
        GIVING TPS-RESULT.
*
*
 PURGETRUSER.
     CALL "PURGETRUSER         OF BANKTR/CODE/HOSTLIB"
        USING IDNUM
        GIVING TPS-RESULT.
*
*
 LOGONTRUSER.
     CALL "LOGONTRUSER         OF BANKTR/CODE/HOSTLIB"
        USING ID-X, IDNUM
        GIVING TPS-RESULT.
*
*
 LOGOFFTRUSER.
     CALL "LOGOFFTRUSER         OF BANKTR/CODE/HOSTLIB"
        USING IDNUM
        GIVING TPS-RESULT.
*
*
 RETURNRESTARTINFO.
     CALL "RETURNRESTARTINFO    OF BANKTR/CODE/HOSTLIB"
        USING IDNUM, TROUT
        GIVING TPS-RESULT.
*
*
 RETURNLASTRESPONSE.
     CALL "RETURNLASTRESPONSE    OF BANKTR/CODE/HOSTLIB"
        USING IDNUM, TROUT
        GIVING TPS-RESULT.
*
*
 TANKTRNORESTART.
     CALL "TANKTRNORESTART      OF BANKTR/CODE/HOSTLIB"
        USING IDNUM, TRIN
        GIVING TPS-RESULT.
*
*
 PROCESSTRANSACTION.
     CALL "PROCESSTRANSACTION      OF BANKTR/CODE/HOSTLIB"
        USING IDNUM, TRIN, TROUT, RESTARTTRREC
        GIVING TPS-RESULT.
*
*
 OPENTRBASE.
```

Example of a Transaction Base

```
     CALL "OPENTRBASE              OF BANKTR/CODE/HOSTLIB"
        USING OPT, TIMEOUTV
        GIVING TPS-RESULT.

*
*
 CLOSETRBASE.
     CALL "CLOSETRBASE             OF BANKTR/CODE/HOSTLIB"
         GIVING TPS-RESULT.
*
*
 SEEKTRANSACTION.
     CALL "SEEKTRANSACTION         OF BANKTR/CODE/HOSTLIB"
         USING FILENUMBER, BLOCKNUMBER, OFFSETS
         GIVING TPS-RESULT.
*
*
 READTRANSACTION.
     CALL "READTRANSACTION         OF BANKTR/CODE/HOSTLIB"
         USING TRIN
         GIVING TPS-RESULT.
*
*
 SWITCHTRFILE.
     CALL "SWITCHTRFILE            OF BANKTR/CODE/HOSTLIB"
          GIVING TPS-RESULT.
*
*
 HANDLESTATISTICS.
     CALL "HANDLESTATISTICS        OF BANKTR/CODE/HOSTLIB"
          USING STATOPTION
          GIVING TPS-RESULT.
*
*****************************************************************
```

ALGOL

```
BEGIN   % SAMPLE BATCH PROGRAM USING TRANSACTIONS
    COMMENT
    THIS PROGRAM CAN RUN ON EITHER A HOST SYSTEM OR A REMOTE
    SYSTEM.  THE LIBRARY ROUTINES, DECLARED BELOW, PROVIDE
    THE PROPER FUNCTION FOR EITHER ENVIRONMENT.
    ;

    ARRAY LIBPARAM[0:9];

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
%  DECLARE THE TRANSACTION BASE TO BE USED
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

   TRANSACTION BASE TRB = BANKTR;     % EXAMPLE OF EQUATING AN INTERNAL
                                       % NAME TO THE TRANSACTION BASE
LIBRARY L(TITLE="BANKTR/CODE/HOSTLIB.");

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
%  DECLARE ALL THE LIBRARY ENTRY POINTS TO BE USED
%
 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

INTEGER PROCEDURE CREATETRUSER(IDSTRING, IDNUM);
    STRING IDSTRING; INTEGER IDNUM;
    LIBRARY L;
INTEGER PROCEDURE PURGETRUSER(IDNUM);
    INTEGER IDNUM;
    LIBRARY L;
INTEGER PROCEDURE LOGONTRUSER(IDSTRING, IDNUM);
    STRING IDSTRING; INTEGER IDNUM;
    LIBRARY L;
INTEGER PROCEDURE LOGOFFTRUSER(IDNUM);
    INTEGER IDNUM;
    LIBRARY L;
INTEGER PROCEDURE RETURNLASTADDRESS(FILENUM, BLOCKNUM, OFFSET, IDNUM);
    INTEGER IDNUM;
    REAL FILENUM, BLOCKNUM, OFFSET;
    LIBRARY L;
INTEGER PROCEDURE RETURNRESTARTINFO(IDNUM, TROUT);
    INTEGER IDNUM;
    TRANSACTION RECORD (TRB) TROUT;
    LIBRARY L;
INTEGER PROCEDURE RETURNLASTRESPONSE(IDNUM, TROUT);
    INTEGER IDNUM;
    TRANSACTION RECORD (TRB) TROUT;
    LIBRARY L;
```

Example of a Transaction Base

```
INTEGER PROCEDURE TANKTRNORESTART(IDNUM, TRIN);
    INTEGER IDNUM;
    TRANSACTION RECORD (TRB) TRIN;
    LIBRARY L;

INTEGER PROCEDURE PROCESSTRANSACTION(IDNUM, TRIN, TROUT, RESTARTTRREC);
    INTEGER IDNUM;
    TRANSACTION RECORD (TRB) TRIN, TROUT, RESTARTTRREC;
    LIBRARY L;
INTEGER PROCEDURE PROCESSTRNORESTART(IDNUM, TRIN, TROUT);
    INTEGER IDNUM;
    TRANSACTION RECORD (TRB) TRIN, TROUT;
    LIBRARY L;
INTEGER PROCEDURE OPENTRBASE(USEROPTION, TIMEOUT);
    INTEGER USEROPTION, TIMEOUT;
    LIBRARY L;
INTEGER PROCEDURE CLOSETRBASE;
    LIBRARY L;
INTEGER PROCEDURE SEEKTRANSACTION(FILENUM, BLOCKNUM, OFFSET);
    INTEGER FILENUM, BLOCKNUM, OFFSET;
    LIBRARY L;
INTEGER PROCEDURE READTRANSACTION (TRREC);
    TRANSACTION RECORD (TRB) TRREC;
    LIBRARY L;
INTEGER PROCEDURE SWITCHTRFILE;
    LIBRARY L;
INTEGER PROCEDURE HANDLESTATISTICS(STATOPTION);
    VALUE STATOPTION; INTEGER STATOPTION;
    LIBRARY L;



    FILE LINE(KIND=PRINTER);
    FILE RMOTE(KIND=REMOTE, MYUSE = IO);
    TRANSACTION RECORD (TRB)
         TRIN,
         TROUT,
         LASTINPUT,
         RESTARTTRREC,
         LASTRESPONSE;
    REAL IDNUM, N, OPT;
    INTEGER ACCT,TIMEOUT,STATISTICSOPTION;
    INTEGER ACCT,TIMEOUT;
    INTEGER RSLT;
    ARRAY SP[0:14];
    BOOLEAN ERROR;
    LABEL EXIT;
    STRING ID, FNAME, JOURNALNAME;

%%%
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DEFINE TANKING = 3#;

DEFINE ERR =
BEGIN
     WRITE(RMOTE, <"RSLT = ", I3, "    ", I8>, RSLT, LINENUMBER);
     ERROR := TRUE;
END#;

DEFINE GETSTRING(S, X) =
BEGIN
    REPLACE SP BY " " FOR 15 WORDS;
    REPLACE SP BY "ENTER STRING FOR ", S;
    WRITE(RMOTE, 15, SP);
    READ(RMOTE, 15, SP);
    SCAN SP FOR N:80 WHILE IN ALPHA;
    X := STRING(SP, 80-N);
END#;

DEFINE GETINTEGER(S, I) =
BEGIN
     REPLACE SP BY " " FOR 15 WORDS;
     REPLACE SP BY "ENTER INTEGER FOR ", S;
     WRITE(RMOTE, 15, SP);
    READ(RMOTE, /, I);
END#;

DEFINE GETREAL(S, R) =
BEGIN
    REPLACE SP BY " " FOR 15 WORDS;
    REPLACE SP BY "ENTER VALUE FOR ", S;
    WRITE(RMOTE, 15, SP);
    READ(RMOTE, /, R);
END#;

DEFINE GETACCT = GETINTEGER("ACCOUNT NUMBER", ACCT)#;

PROCEDURE PROCESSTR;
BEGIN
   IF OPT = TANKING THEN
   BEGIN
      IF RSLT := TANKTRNORESTART(IDNUM, TRIN) > 0 THEN ERR;
   END
   ELSE
      IF RSLT := PROCESSTRANSACTION(IDNUM, TRIN, TROUT, RESTARTTRREC)
            > 0 THEN ERR;
END PROCESSTR;

PROCEDURE GETLASTP;
BEGIN
   IF OPT = TANKING THEN
   BEGIN
```

```
      IF RSLT := RETURNRESTARTINFO(IDNUM, LASTINPUT) > 0 THEN
         ERR
      ELSE
         WRITE(RMOTE,
              <"LAST RESTART  (FILE, BLOCK, OFFSET, FORMAT): ", 4I5>,
              LASTINPUT.TRFILENUM,
              LASTINPUT.TRBLOCKNUM, LASTINPUT.TROFFSET,
              LASTINPUT.TRFORMAT );
   END ELSE
   BEGIN
      IF RSLT := RETURNRESTARTINFO(IDNUM, LASTINPUT) > 0 THEN
         ERR
      ELSE
      BEGIN
         WRITE(RMOTE,
             <"LAST RESTART  (FILE, BLOCK, OFFSET, FORMAT): ", 4I5>,
             LASTINPUT.TRFILENUM,
             LASTINPUT.TRBLOCKNUM, LASTINPUT.TROFFSET,
             LASTINPUT.TRFORMAT );
         IF RSLT := RETURNLASTRESPONSE(IDNUM, LASTRESPONSE) > 0 THEN
            ERR
         ELSE
            WRITE(RMOTE,
                <"LAST RESPONSE (FILE, BLOCK, OFFSET, FORMAT): ", 4I5>,
                 LASTRESPONSE.TRFILENUM,
                 LASTRESPONSE.TRBLOCKNUM, LASTRESPONSE.TROFFSET,
                 LASTRESPONSE.TRFORMAT );
      END;
   END;
END GETLASTP;


PROCEDURE DISPLAYSTATUS;
BEGIN
    IF TROUT.TRFORMAT NEQ TRFORMAT(STATUS) THEN
       ERR
    ELSE
       WRITE(RMOTE, <"ACCOUNT NUMBER ", I5,
            ": CURRENT BALANCE IS ", F10.2>,
            TROUT.STATUS.ACCTNUM,
            TROUT.STATUS.BALANCE);
END DISPLAYSTATUS;


PROCEDURE CREATEP;   % CREATE A NEW ACCOUNT NUMBER
BEGIN
    STRING NAME;
    WRITE(RMOTE, <"FUNCTION IS CREATE">);
    GETACCT;
    GETSTRING("CUSTOMER NAME", NAME);
    CREATE TRIN.CREATEACCT;
    TRIN.CREATEACCT.ACCTNUM := ACCT;
    TRIN.CREATEACCT.NAME := NAME;
```

```
    PROCESSTR;
END CREATEP;

PROCEDURE PURGEP;    % ELIMINATE AN ACCOUNT NUMBER
BEGIN
    WRITE(RMOTE, <"FUNCTION IS PURGE">);
    GETACCT;
    CREATE TRIN.PURGEACCT;
    TRIN.PURGEACCT.ACCTNUM := ACCT;
    PROCESSTR;
END PURGEP;


PROCEDURE STATUSP;   % DISPLAY THE STATUS OF AN ACCOUNT
BEGIN
    WRITE(RMOTE, <"FUNCTION IS STATUS">);
    GETACCT;
    CREATE TRIN.STATUS;
    TRIN.STATUS.ACCTNUM := ACCT;
    PROCESSTR;
    IF (OPT NEQ TANKING AND NOT ERROR) THEN DISPLAYSTATUS;
END STATUSP;


PROCEDURE DEPOSITP;    % DEPOSIT SOME AMOUNT IN AN ACCOUNT
BEGIN
    REAL AMT;
    WRITE(RMOTE, <"FUNCTION IS DEPOSIT">);
    GETACCT;
    GETREAL("AMOUNT OF DEPOSIT", AMT);
    CREATE TRIN.DEPOSIT;
    TRIN.DEPOSIT.ACCTNUM := ACCT;
    TRIN.DEPOSIT.TRANDATE := TIME(6);
    TRIN.DEPOSIT.AMOUNT := AMT;
    PROCESSTR;
    IF (OPT NEQ TANKING AND NOT ERROR) THEN DISPLAYSTATUS;
END DEPOSITP;

PROCEDURE WITHDRAWALP;    % WITHDRAWL SOME AMOUNT FROM AN ACCOUNT
BEGIN
    REAL AMT;
    WRITE(RMOTE, <"FUNCTION IS WITHDRAWAL">);
    GETACCT;
    GETREAL("AMOUNT OF WITHDRAWAL", AMT);
    CREATE TRIN.WITHDRAWAL;
    TRIN.WITHDRAWAL.ACCTNUM := ACCT;
    TRIN.WITHDRAWAL.TRANDATE := TIME(6);
    TRIN.WITHDRAWAL.AMOUNT := AMT;
    PROCESSTR;
    IF (OPT NEQ TANKING AND NOT ERROR) THEN DISPLAYSTATUS;
END WITHDRAWALP;

PROCEDURE NEWUSERP;
```

```
BEGIN
    WRITE(RMOTE, <"FUNCTION IS NEWUSER">);
    GETSTRING("USER ID", ID);
    WRITE(RMOTE, <"USER: ", A15>, ID);
    IF RSLT := LOGONTRUSER(ID, IDNUM) > 0 THEN
       ERR
    ELSE
       WRITE(RMOTE, <"USER #: ", I3>, IDNUM);
END NEWUSERP;

PROCEDURE REOPENP;
BEGIN
    WRITE(RMOTE, <"FUNCTION IS REOPEN">);
    IF RSLT := CLOSETRBASE > 0 THEN
       ERR
    ELSE
    BEGIN
       WRITE(RMOTE, <"WHAT DO YOU WANT TO DO?">);
       GETINTEGER("CHOICE (1=UPDATE, 2=INQUIRY, 3=TANK, 4=READ,"
                  " 5=EXCLUSIVEUPDATE)", OPT);
       IF RSLT := OPENTRBASE(OPT, 0) > 0 THEN ERR;
    END;
END REOPENP;

PROCEDURE SEEKP;
BEGIN
    REAL FILENUM, BLOCKNUM, OFFSET;
    WRITE(RMOTE, <"FUNCTION IS SEEK">);
    WRITE(RMOTE, <"ENTER FILENUM, BLOCKNUM, OFFSET">);
    READ(RMOTE, /, FILENUM, BLOCKNUM, OFFSET);
    IF RSLT := SEEKTRANSACTION(FILENUM, BLOCKNUM, OFFSET) > 0 THEN ERR;
END SEEKP;

PROCEDURE READP;
BEGIN
    WRITE(RMOTE, <"FUNCTION IS READ">);
    IF RSLT := READTRANSACTION(TRIN) > 0 THEN
       ERR
    ELSE
    BEGIN
       WRITE(RMOTE, <"FILE, BLOCK, OFFSET:", 3I5>,
         TRIN.TRFILENUM,
         TRIN.TRBLOCKNUM,
         TRIN.TROFFSET );
       WRITE(RMOTE, <"FORMAT, SUBFORMAT:", 2I5>,
         TRIN.TRFORMAT,
         TRIN.TRSUBFORMAT );
    END;
END READP;

PROCEDURE CREATEUSERP;
BEGIN
    WRITE(RMOTE, <"FUNCTION IS CREATEUSER">);
```

```
    GETSTRING("USER ID", ID);
    WRITE(RMOTE, <"USER: ", A15>, ID);
    IF RSLT := CREATETRUSER(ID, IDNUM) > 0 THEN
       ERR
    ELSE
       IF RSLT := LOGONTRUSER(ID, IDNUM) > 0 THEN
          ERR
       ELSE
          WRITE(RMOTE, <"USER #: ", I3>, IDNUM);
END CREATEUSERP;

PROCEDURE PURGEUSERP;
BEGIN
    WRITE(RMOTE, <"FUNCTION IS PURGEUSER">);
    IF RSLT := PURGETRUSER(IDNUM) > 0 THEN ERR;
END PURGEUSERP;
PROCEDURE QUITP;
BEGIN
    WRITE(RMOTE, <"FUNCTION IS QUIT">);
    CLOSETRBASE;
    GO EXIT;
END QUITP;

PROCEDURE SWITCHP;
BEGIN
    WRITE(RMOTE, <"FUNCTION IS SWITCH">);
    IF RSLT := SWITCHTRFILE > 0 THEN ERR;
END SWITCHP;

PROCEDURE STATISTICSP;
BEGIN
    WRITE(RMOTE, <"FUNCTION IS STATISTICS">);
    WRITE(RMOTE, <"WHAT DO YOU WANT TO DO?");
    GETINTEGER("CHOICE (1 = PRINT & RESET, 2 = PRINT"
               "ONLY, 3 = RESET)", STATISTICSOPTION);
    IF RSLT := HANDLESTATISTICS(STATISTICSOPTION) > 0;
    THEN ERR;
END STATISTICSP;

PROCEDURE HELPP;
BEGIN
    WRITE(RMOTE, <"FUNCTIONS ARE:", /,
         "CREATE", /,
         "PURGE", /,
         "DEPOSIT", /,
         "WITHDRAWAL", /,
         "QUIT", /,
         "STATUS", /,
         "NEWUSER", /,
         "REOPEN", /,
         "SEEK", /,
         "READ", /,
         "GETLAST", /,
```

```
          "CREATEUSER", /,
          "PURGEUSER", /,
          "SWITCH", /,
          "STATISTICS", /,
          "HELP" > );
END HELPP;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% SET LIBPARAMETER BEFORE FIRST CALL ON A LIBRARY ENTRYPOINT. ONE COULD
% HAVE SET LIBPARAMETER IN THE LIBRARY DECLARATION RATHER THAN HERE.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    GETSTRING("JOURNAL NAME", JOURNALNAME);
    L.LIBPARAMETER := JOURNALNAME;

    WRITE(RMOTE, <"WHAT DO YOU WANT TO DO?">);
    GETINTEGER("CHOICE (1=UPDATE, 2=INQUIRY, 3=TANK, 4=READ,"
               " 5=EXCLUSIVEUPDATE)", OPT);
    WRITE(RMOTE,<"WHAT VALUE FOR TIMEOUT SHALL WE USE?">);
    READ(RMOTE,/,TIMEOUT);
    IF RSLT := OPENTRBASE(OPT, TIMEOUT) > 0 THEN ERR;
    %
    %  HERE WE CREATE A RESTART TRANSACTION RECORD TO BE WRITTEN TO THE
    %  TRHISTORY FILE ALONG WITH AN INPUT TRANSACTION. NOTE THAT WE DO
    %  NOT ASSIGN VALUES TO THE ITEMS OF THIS RECORD. NORMALLY ONE
    %  WOULD DO SO, BUT FOR SIMPLICITY WE LEFT THAT CODE OUT OF THIS
    %  EXAMPLE.
    %
    IF NOT ERROR THEN
    BEGIN
       CREATE RESTARTTRREC.RESTARTDETANKER;
       GETSTRING("USER ID", ID);
       WRITE(RMOTE, <"USER: ", A15>, ID);
       IF RSLT := LOGONTRUSER(ID, IDNUM) > 0 THEN ERR;
       IF NOT ERROR THEN
          WRITE(RMOTE, <"USER #: ", I3>, IDNUM);
    END;
    ERROR := FALSE;
    WHILE TRUE DO
    BEGIN
       GETSTRING("FUNCTION NAME (OR HELP)", FNAME);
       IF SP = "CREATEUSER" THEN CREATEUSERP ELSE
       IF SP = "PURGEUSER" THEN PURGEUSERP ELSE
       IF SP = "CREATE" THEN CREATEP ELSE
       IF SP = "PURGE" THEN PURGEP ELSE
       IF SP = "DEPOSIT" THEN DEPOSITP ELSE
       IF SP = "WITHDRAWAL" THEN WITHDRAWALP ELSE
       IF SP = "QUIT" THEN QUITP ELSE
       IF SP = "STATUS" THEN STATUSP ELSE
       IF SP = "NEWUSER" THEN NEWUSERP ELSE
       IF SP = "REOPEN" THEN REOPENP ELSE
```

```
        IF SP = "SEEK" THEN SEEKP ELSE
        IF SP = "R" THEN READP ELSE
        IF SP = "HELP" THEN HELPP ELSE
        IF SP = "GETLAST" THEN GETLASTP ELSE
        IF SP = "SW" THEN SWITCHP ELSE
        IF SP = "STAT" THEN STATISTICSP ELSE
        WRITE(RMOTE, <"DID NOT RECOGNIZE FUNCTION NAME">);
        ERROR := FALSE;
    END;

EXIT:

END OF THE APPLICATION PROGRAM.
```

EXAMPLES OF UPDATE LIBRARY

The following programs written in COBOL and ALGOL are Update Libraries capable of maintaining data base consistency and ensuring reproducibility. Refer to Section 6 of this manual for more information about programming Update Libraries, and for an additional example.

COBOL

```
$SET SHARING = PRIVATE
$SET TEMPORARY
 IDENTIFICATION DIVISION.
 PROGRAM-ID. ACCESSDATABASE.
 ENVIRONMENT DIVISION.
 DATA DIVISION.
 DATA-BASE SECTION.
 DB DBASE = BANKDB ALL.
 TRANSACTION SECTION.
 TB  TRB = BANKTR
     USING STATIS = STATUS, CREATEACCT, RESTARTDETANKER, DEPOSIT,
           WITHDRAWAL, PURGEACCT, SYSTEMTR.
 01  TRIN.
 01  TROUT.
*
 WORKING-STORAGE SECTION.
 77  FUNCTIONFLAG    COMP    PIC 9(8).
 77  ACCT            COMP    PIC 9(6).
 77  OLDBAL          COMP    PIC S9(8)V99.
 77  NEWBAL          COMP    PIC S9(8)V99.
*
**********************************************************************
*
 PROCEDURE DIVISION USING FUNCTIONFLAG,TRIN,TROUT,SAVEINPUT,
           SAVERESPONSE.
 DECLARATIVES.
    SAVEINPUT SECTION. USE AS TRANSACTION PROCEDURE.
    SAVERESPONSE SECTION. USE AS TRANSACTION PROCEDURE.
 END DECLARATIVES.
*
 MAIN SECTION.
 P1.
     IF FUNCTIONFLAG EQUAL 1  PERFORM OPENUPDATE     ELSE
     IF FUNCTIONFLAG EQUAL 2  PERFORM OPENINQUIRY    ELSE
     IF FUNCTIONFLAG EQUAL 3  PERFORM UPDATEDB       ELSE
     IF FUNCTIONFLAG EQUAL 4  PERFORM FORCEABORT     ELSE
     IF FUNCTIONFLAG EQUAL 5  PERFORM CLOSEDB.
 UPDATELIB-EXIT.
     EXIT PROGRAM.
*
```

```
**********************************************************************
*
 OPENUPDATE.
***********
      OPEN TRUPDATE DBASE.

*
*
 OPENINQUIRY.
************
      OPEN INQUIRY DBASE.
*
*
 UPDATEDB.
*********
      IF TRFORMAT(TRIN) EQUAL TRFORMAT(CREATEACCT) PERFORM MAKEACCT
                                                           ELSE
      IF TRFORMAT(TRIN) EQUAL TRFORMAT(PURGEACCT)  PERFORM ELIMACCT
                                                           ELSE
      IF TRFORMAT(TRIN) EQUAL TRFORMAT(STATIS)     PERFORM STATUSCHK
                                                           ELSE
      IF TRFORMAT(TRIN) EQUAL TRFORMAT(DEPOSIT)    PERFORM DEPAMNT
                                                           ELSE
      IF TRFORMAT(TRIN) EQUAL TRFORMAT(WITHDRAWAL) PERFORM WITHAMNT
                                                           ELSE
      DISPLAY "NO UPDATE ROUTINE FOR THE FORMAT PASSED IN" .
*
*
 FORCEABORT.
***********
      CLOSE DBASE.
*
*
 CLOSEDB.
********
      CLOSE DBASE.
*
*
*  THIS IS THE UPDATE ROUTINE FOR CREATING A NEW ACCOUNT.
*
 MAKEACCT.
*********
      CREATE ACCOUNT ON EXCEPTION PERFORM ERR.
      MOVE ACCTNUM OF CREATEACCT OF TRIN TO ACCOUNT-NUM.
      MOVE NAME OF CREATEACCT OF TRIN TO NAME OF ACCOUNT.
      BEGIN-TRANSACTION(TRIN) RDS ON EXCEPTION PERFORM ERR.
      MID-TRANSACTION(TRIN,SAVEINPUT) RDS ON EXCEPTION PERFORM ERR.
      STORE ACCOUNT ON EXCEPTION PERFORM ERR.
*
*  RETURN INPUT TRANSACTION AS GOOD TRANSACTION RESPONSE RECORD.
*
      MOVE TRIN TO TROUT.
      END-TRANSACTION(TRIN,SAVERESPONSE) RDS ON EXCEPTION PERFORM
```

```
          ERR.
*
*
*  THIS IS THE UPDATE ROUTINE FOR ELIMINATING AN EXISTING ACCOUNT.
*
 ELIMACCT.
*********
     MOVE ACCTNUM OF PURGEACCT OF TRIN TO ACCT.
     LOCK ACCOUNT-SET AT ACCOUNT-NUM = ACCT ON EXCEPTION PERFORM
        ERR.
     BEGIN-TRANSACTION(TRIN) RDS ON EXCEPTION PERFORM ERR.
     MID-TRANSACTION(TRIN,SAVEINPUT) RDS ON EXCEPTION PERFORM ERR.
     DELETE ACCOUNT ON EXCEPTION PERFORM ERR.
*
*  RETURN SAME TRANSACTION AS GOOD TRANSACTION RESPONSE RECORD.
*
     MOVE TRIN TO TROUT.
     END-TRANSACTION(TRIN,SAVERESPONSE) RDS ON EXCEPTION PERFORM
        ERR.
*
*
*  THIS IS AN EXAMPLE OF AN INQUIRY ROUTINE. IT RETURNS THE BALANCE
*  OF A PARTICULAR ACCOUNT.
*
 STATUSCHK.
**********
     MOVE ACCTNUM OF STATIS OF TRIN TO ACCT.
     FIND ACCOUNT-SET AT ACCOUNT-NUM = ACCT ON EXCEPTION PERFORM
        ERR.
     MOVE TRIN TO TROUT.
     MOVE BALANCE OF ACCOUNT TO BALANCE OF STATIS OF TROUT.
*
*
*  THIS IS THE UPDATE ROUTINE TO PERFORM DEPOSITS INTO ACCOUNTS.
*
 DEPAMNT.
********
     MOVE ACCTNUM OF DEPOSIT OF TRIN TO ACCT.
     LOCK ACCOUNT-SET AT ACCOUNT-NUM = ACCT ON EXCEPTION PERFORM
        ERR.
     MOVE BALANCE OF ACCOUNT TO OLDBAL.
     ADD OLDBAL, AMOUNT OF DEPOSIT OF TRIN GIVING NEWBAL.
     CREATE DEPOSIT OF ACCOUNT ON EXCEPTION PERFORM ERR.
     MOVE TRANDATE OF DEPOSIT OF TRIN TO TRANDATE OF DEPOSIT OF
        ACCOUNT.
     MOVE AMOUNT OF DEPOSIT OF TRIN TO AMOUNT OF DEPOSIT OF ACCOUNT.
     MOVE OLDBAL TO OLD-BALANCE.
     MOVE NEWBAL TO NEW-BALANCE.
     MOVE NEWBAL TO BALANCE OF ACCOUNT.
     BEGIN-TRANSACTION(TRIN) RDS ON EXCEPTION PERFORM ERR.
     MID-TRANSACTION(TRIN,SAVEINPUT) RDS ON EXCEPTION PERFORM ERR.
     STORE ACCOUNT ON EXCEPTION PERFORM ERR.
```

```
      STORE DEPOSIT OF ACCOUNT ON EXCEPTION PERFORM ERR.
      CREATE STATIS IN TROUT.
      MOVE NEWBAL TO BALANCE OF STATIS OF TROUT.
      MOVE ACCTNUM OF DEPOSIT OF TRIN TO ACCTNUM OF STATIS OF TROUT.
      END-TRANSACTION(TRIN,SAVERESPONSE) RDS ON EXCEPTION PERFORM
         ERR.

*
*
*  THIS IS THE UPDATE ROUTINE THAT PERFORMS WITHDRAWALS AGAINST
*  ACCOUNTS.
*
 WITHAMNT.
*********
      MOVE ACCTNUM OF WITHDRAWAL OF TRIN TO ACCT.
      LOCK ACCOUNT-SET AT ACCOUNT-NUM = ACCT ON EXCEPTION PERFORM
         ERR.
      MOVE BALANCE OF ACCOUNT TO OLDBAL.
      SUBTRACT AMOUNT OF WITHDRAWAL OF TRIN FROM OLDBAL GIVING
         NEWBAL.
      CREATE DEPOSIT OF ACCOUNT ON EXCEPTION PERFORM ERR.
      MOVE TRANDATE OF WITHDRAWAL OF TRIN TO TRANDATE OF DEPOSIT OF
         ACCOUNT.
      MOVE AMOUNT OF WITHDRAWAL OF TRIN TO AMOUNT OF DEPOSIT OF
         ACCOUNT.
      MOVE OLDBAL TO OLD-BALANCE.
      MOVE NEWBAL TO NEW-BALANCE.
      MOVE NEWBAL TO BALANCE OF ACCOUNT.
      BEGIN-TRANSACTION(TRIN) RDS ON EXCEPTION PERFORM ERR.
      MID-TRANSACTION(TRIN,SAVEINPUT) RDS ON EXCEPTION PERFORM ERR.
      STORE ACCOUNT ON EXCEPTION PERFORM ERR.
      STORE DEPOSIT OF ACCOUNT ON EXCEPTION PERFORM ERR.
      CREATE STATIS IN TROUT.
      MOVE NEWBAL TO BALANCE OF STATIS OF TROUT.
      MOVE ACCTNUM OF DEPOSIT OF TRIN TO ACCTNUM OF STATIS OF TROUT.
      END-TRANSACTION(TRIN,SAVERESPONSE) RDS ON EXCEPTION PERFORM
         ERR.
*
*
*  REPORT ANY EXCEPTIONS THAT WERE DETECTED.
*
 ERR.
*****
      DISPLAY "UPDATE RESULT:".
      DISPLAY "CATEGORY, ERRORTYPE, STRUCTURE " DMSTATUS(DMCATEGORY)
              ", " DMSTATUS(DMERRORTYPE) ", " DMSTATUS(DMSTRUCTURE).
      GO TO UPDATELIB-EXIT.
```

ALGOL

```
% REMEMBER: ONE MUST SET SHARING = PRIVATE IN ORDER TO ENSURE THAT EACH
%           APPLICATION PROGRAM WILL GET ITS OWN COPY OF THE UPDATE
%           LIBRARY.
$SET SHARING=PRIVATE
BEGIN  % USER'S TRANSACTION UPDATE LIBRARY

    COMMENT:
     *********
       THIS LIBRARY IS WRITTEN BY THE USER OF THE TRANSACTION
       SYSTEM.  IT CONSISTS OF A SINGLE PROCEDURE,
       CALLED "ACCESSDATABASE", WHICH IS DESIGNED
       TO PERFORM FOUR BASIC FUNCTIONS:
       OPENDATABASE( FOR UPDATE OR INQUIRY), UPDATE, FORCEABORT, AND
       CLOSEDATABASE.
       THE FUNCTION TO BE PERFORMED IS IDENTIFIED BY THE
       FIRST PARAMETER TO THE PROCEDURE.

       OPENDATABASE(FOR UPDATE OR INQUIRY) IS REQUIRED TO OPEN THE DATA
       BASE.

       UPDATE IS CALLED BY THE TRANSACTION SYSTEM
       ONCE FOR EACH INPUT TRANSACTION TO BE PROCESSED.  IT MUST
       OBSERVE A FEW SIMPLE RULES, SUCH AS WHEN TO LOCK RECORDS,
       WHEN TO CALL THE FORMAL PROCEDURES, ETC.  IT IS EXPECTED
       TO EXAMINE EACH INPUT TRANSACTION RECORD, PERFORM THE
       APPROPRIATE ACTIONS, CREATE A RESPONSE TRANSACTION, AND EXIT.

       FORCEABORT IS REQUIRED SO THAT THE TRANSACTION SYSTEM
       MAY CAUSE AN ABORT IF NECESSARY.

       CLOSEDATABASE MUST CLOSE THE DATABASE.

    ;
%*********************************************************************
%
%    LIBRARY GLOBAL DECLARATIONS
%
%*********************************************************************


   DATABASE DB = BANKDB;          % INVOKE THE DATA BASE AND TRANSACTION
                                  % BASE TO BE USED.
   TRANSACTION BASE TRB = BANKTR;
   EBCDIC ARRAY SPO[0:79];


%*********************************************************************
%
%    PROCEDURE UPDATE
```

```
%
%*************************************************************************

PROCEDURE UPDATE(TRIN, TROUT, SAVEINPUT, SAVERESPONSE);
    TRANSACTION RECORD (TRB) TRIN, TROUT;
    PROCEDURE SAVERESPONSE(); FORMAL;
    PROCEDURE SAVEINPUT(); FORMAL;
BEGIN

    LABEL EXIT;
    BOOLEAN RSLT;
    CASE TRIN.TRFORMAT OF
    BEGIN

    (TRFORMAT(CREATEACCT)):      % ROUTINE FOR CREATING A NEW ACCOUNT
    BEGIN
         STRING SNAME;
         EBCDIC ARRAY NAME[0:29];
         SNAME := TRIN.CREATEACCT.NAME;
         REPLACE NAME[0] BY SNAME;
         CREATE ACCOUNT:RSLT;
         IF RSLT THEN GO EXIT;
         PUT ACCOUNT
         (
              ACCOUNT-NUM := TRIN.CREATEACCT.ACCTNUM,
              NAME := NAME[0];
         );
         BEGINTRANSACTION (TRIN) RDS :RSLT;
         IF RSLT THEN GO EXIT;
         MIDTRANSACTION (TRIN, SAVEINPUT) RDS: RSLT;
         IF RSLT THEN GO EXIT;
         STORE ACCOUNT:RSLT;
         IF RSLT THEN GO EXIT;
         TROUT := TRIN;  % RETURN SAME TR AS GOOD TR-RESPONSE
         ENDTRANSACTION (TRIN, SAVERESPONSE) RDS :RSLT;
         IF RSLT THEN GO EXIT;
    END CREATEACCT FORMAT;

    (TRFORMAT(PURGEACCT)):       % ROUTINE FOR PURGING AN EXISTING
                                 % ACCOUNT.
    BEGIN
         REAL ACCT;
         ACCT := TRIN.PURGEACCT.ACCTNUM;
         LOCK ACCOUNT-SET AT ACCOUNT-NUM = ACCT :RSLT;
         IF RSLT THEN GO EXIT;
         BEGINTRANSACTION (TRIN) RDS :RSLT;
         IF RSLT THEN GO EXIT;
         MIDTRANSACTION (TRIN, SAVEINPUT) RDS: RSLT;
         IF RSLT THEN GO EXIT;
         DELETE ACCOUNT:RSLT;
         IF RSLT THEN GO EXIT;
         TROUT := TRIN;  % SIGNAL OK
         ENDTRANSACTION (TRIN, SAVERESPONSE) RDS :RSLT;
```

```
     IF RSLT THEN GO EXIT;
END PURGEACCT FORMAT;

(TRFORMAT(STATUS)):       % AN EXAMPLE OF AN INQUIRY ROUTINE. IT
BEGIN                     % RETURNS THE BALANCE OF A PARTICULAR
                          % ACCOUNT.
     REAL ACCT, BAL;
     ACCT := TRIN.STATUS.ACCTNUM;
     FIND ACCOUNT-SET AT ACCOUNT-NUM = ACCT :RSLT;
     IF RSLT THEN GO EXIT;
     GET ACCOUNT
     (
          BAL := BALANCE
     );
     TROUT := TRIN;  % SIGNAL OK
     TROUT.STATUS.BALANCE := BAL;
END STATUS FORMAT;

(TRFORMAT(DEPOSIT)):      % ROUTINE TO PERFORM A DEPOSIT INTO AN
BEGIN                     % ACCOUNT.
     REAL OLDBAL, NEWBAL;
     REAL ACCT;
     ACCT := TRIN.DEPOSIT.ACCTNUM;
     LOCK ACCOUNT-SET AT ACCOUNT-NUM = ACCT :RSLT;
     IF RSLT THEN GO EXIT;
     GET ACCOUNT
     (
          OLDBAL := BALANCE
     );
     NEWBAL := OLDBAL + TRIN.DEPOSIT.AMOUNT;
     CREATE DEPOSIT:RSLT;
     IF RSLT THEN GO EXIT;
     PUT DEPOSIT
     (
          TRANDATE := TRIN.DEPOSIT.TRANDATE,
          AMOUNT := TRIN.DEPOSIT.AMOUNT,
          OLD-BALANCE := OLDBAL,
          NEW-BALANCE := NEWBAL
     );
     PUT ACCOUNT
     (
          BALANCE := NEWBAL
     );
     BEGINTRANSACTION (TRIN) RDS :RSLT;
     IF RSLT THEN GO EXIT;
     MIDTRANSACTION (TRIN, SAVEINPUT) RDS: RSLT;
     IF RSLT THEN GO EXIT;
     STORE ACCOUNT:RSLT;
     IF RSLT THEN GO EXIT;
     STORE DEPOSIT:RSLT;
     IF RSLT THEN GO EXIT;
     CREATE TROUT.STATUS;
     TROUT.STATUS.BALANCE := NEWBAL;
```

```
        TROUT.STATUS.ACCTNUM := TRIN.DEPOSIT.ACCTNUM;
        ENDTRANSACTION (TRIN, SAVERESPONSE) RDS :RSLT;
        IF RSLT THEN GO EXIT;
    END DEPOSIT FORMAT;

    (TRFORMAT(WITHDRAWAL)):      % ROUTINE TO WITHDRAWAL MONEY FROM AN
                                 % ACCOUNT.
    BEGIN  % USES DEPOSIT DATA SET, NOT WITHDRAWAL
        REAL OLDBAL, NEWBAL;
        REAL ACCT;
        ACCT := TRIN.WITHDRAWAL.ACCTNUM;
        LOCK ACCOUNT-SET AT ACCOUNT-NUM = ACCT :RSLT;
        IF RSLT THEN GO EXIT;
        GET ACCOUNT
        (
            OLDBAL := BALANCE
        );
        NEWBAL := OLDBAL - TRIN.WITHDRAWAL.AMOUNT;
        CREATE DEPOSIT:RSLT;
        IF RSLT THEN GO EXIT;
        PUT DEPOSIT
        (
            TRANDATE := TRIN.WITHDRAWAL.TRANDATE,
            AMOUNT := - TRIN.WITHDRAWAL.AMOUNT,
            OLD-BALANCE := OLDBAL,
            NEW-BALANCE := NEWBAL
        );
        PUT ACCOUNT
        (
            BALANCE := NEWBAL
        );
        BEGINTRANSACTION (TRIN) RDS :RSLT;
        IF RSLT THEN GO EXIT;
        MIDTRANSACTION (TRIN, SAVEINPUT) RDS: RSLT;
        IF RSLT THEN GO EXIT;
        STORE ACCOUNT:RSLT;
        IF RSLT THEN GO EXIT;
        STORE DEPOSIT:RSLT;
        IF RSLT THEN GO EXIT;
        CREATE TROUT.STATUS;
        TROUT.STATUS.BALANCE := NEWBAL;
        TROUT.STATUS.ACCTNUM := TRIN.DEPOSIT.ACCTNUM;
        ENDTRANSACTION (TRIN, SAVERESPONSE) RDS :RSLT;
        IF RSLT THEN GO EXIT;
    END WITHDRAWAL FORMAT;


    ELSE:    % FLAG AN ERROR
        DISPLAY("NO UPDATE ROUTINE FOR THE FORMAT PASSED IN");

    END CASES;
EXIT:
    IF REAL(RSLT) ISNT 0 THEN
```

```
    BEGIN
         REPLACE SPO BY 0 FOR 10 WORDS;
         WRITE(SPO[*], <"UPDATE RSLT:", H13>, RSLT);
         DISPLAY(SPO);
    END;
END UPDATE;

 %************************************************************************
%
%      PROCEDURE ACCESSDATABASE
%
 %************************************************************************

PROCEDURE ACCESSDATABASE(FUNCTIONFLAG, TRIN, TROUT,
         SAVEINPUT, SAVERESPONSE);
    VALUE FUNCTIONFLAG;
    INTEGER FUNCTIONFLAG;
    TRANSACTION RECORD (TRB) TRIN, TROUT;
    PROCEDURE SAVERESPONSE(); FORMAL;
    PROCEDURE SAVEINPUT(); FORMAL;
BEGIN
    CASE FUNCTIONFLAG OF
    BEGIN
         1:                    % OPEN UPDATE
              OPEN TRUPDATE DB;
         2:                    % OPEN INQUIRY
              OPEN INQUIRY DB;
         3:                    % UPDATE
              UPDATE(TRIN, TROUT, SAVEINPUT, SAVERESPONSE);
         4:                    % FORCEABORT. IT IS CALLED BY THE TRANS.
              CLOSE DB;        % LIBRARY WHEN THE LAST CALL RESULTED IN
                               % EXITING THIS LIBRARY WHILE STILL IN
                               % TRANSACTION STATE.
         5:                    % CLOSE
              CLOSE DB;
    END CASES;
END ACCESSDATABASE;



 %************************************************************************
%
%      INITIALIZE LIBRARY
%
 %************************************************************************

    EXPORT
         ACCESSDATABASE;

    FREEZE(TEMPORARY);


END UPDATE LIBRARY.
```

EXAMPLE OF DETANKING PROCEDURE

The following procedure illustrates detanking. The input parameter is the name of the tank journal. This procedure will open both the TANK journal and the TRHISTORY journal, and then read transactions from the Tank journal and process them against the data base. The Transaction Base invoked by this procedure is the same as that for the Banking Example. In addition, this procedure uses the previously defined UPDATE LIBRARY and DASDL description.

ALGOL

```
PROCEDURE DETANKER(TANKNAME);
ARRAY TANKNAME[*];
BEGIN
    COMMENT:
     *********
       THIS PROGRAM CAN RUN ON EITHER THE HOST SYSTEM OR A REMOTE
       SYSTEM.  THE LIBRARY ROUTINES, DECLARED BELOW, PROVIDE
       THE PROPER FUNCTION FOR EITHER ENVIRONMENT.
    ;

STRING TANKLIBPARAM;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
%   DECLARE THE TRANSACTION BASE TO BE USED
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

TRANSACTION BASE TRB = BANKTR;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
%   DECLARE ALL THE LIBRARY ENTRY POINTS TO BE ASSOCIATED WITH THE
%   TRHISTORY JOURNAL
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

LIBRARY PROCESSLIB(TITLE="BANKTR/CODE/HOSTLIB.",
    LIBPARAMETER = "TRHISTORY" );

INTEGER PROCEDURE CREATETRHISTORYUSER(IDSTRING, IDNUM);
    STRING IDSTRING; INTEGER IDNUM;
    LIBRARY PROCESSLIB(ACTUALNAME = "CREATETRUSER");

INTEGER PROCEDURE LOGONTRHISTORYUSER(IDSTRING, IDNUM);
    STRING IDSTRING; INTEGER IDNUM;
    LIBRARY PROCESSLIB (ACTUALNAME = "LOGONTRUSER");
```

```
INTEGER PROCEDURE RETURNRESTARTINFO(IDNUM, TROUT);
    INTEGER IDNUM;
    TRANSACTION RECORD (TRB) TROUT;
    LIBRARY PROCESSLIB;

INTEGER PROCEDURE RETURNLASTRESPONSE(IDNUM, TROUT);
    INTEGER IDNUM;
    TRANSACTION RECORD (TRB) TROUT;
    LIBRARY PROCESSLIB;

INTEGER PROCEDURE PROCESSTRFROMTANK(IDNUM, TRIN, RESTARTNUM, RESTARTTR);
    INTEGER IDNUM, RESTARTNUM;
    TRANSACTION RECORD (TRB) TRIN, RESTARTTR;
    LIBRARY PROCESSLIB;

INTEGER PROCEDURE OPENTRHISTORY(USEROPTION, TIMEOUT);
    INTEGER USEROPTION, TIMEOUT;
    LIBRARY PROCESSLIB(ACTUALNAME = "OPENTRBASE");

INTEGER PROCEDURE CLOSETRHISTORY;
    LIBRARY PROCESSLIB(ACTUALNAME = "CLOSETRBASE");

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
%   DECLARE ALL THE LIBRARY ENTRY POINTS TO BE ASSOCIATED WITH THE
%   TANK JOURNAL
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

LIBRARY TANKLIB(TITLE="BANKTR/CODE/HOSTLIB.");

INTEGER PROCEDURE CREATETANKUSER(IDSTRING, IDNUM);
    STRING IDSTRING; INTEGER IDNUM;
    LIBRARY TANKLIB (ACTUALNAME = "CREATETRUSER");

INTEGER PROCEDURE LOGONTANKUSER(IDSTRING, IDNUM);
    STRING IDSTRING; INTEGER IDNUM;
    LIBRARY TANKLIB(ACTUALNAME = "LOGONTRUSER");

INTEGER PROCEDURE TANKUSERIDSTRING(IDSTRING, IDNUM);
    STRING IDSTRING; INTEGER IDNUM;
    LIBRARY TANKLIB(ACTUALNAME = "TRUSERIDSTRING");

INTEGER PROCEDURE TANKTRNORESTART(IDNUM, TRIN);
    INTEGER IDNUM;
    TRANSACTION RECORD (TRB) TRIN;
    LIBRARY TANKLIB;

INTEGER PROCEDURE OPENTANK(USEROPTION, TIMEOUT);
    INTEGER USEROPTION, TIMEOUT;
    LIBRARY TANKLIB(ACTUALNAME = "OPENTRBASE");
```

```
INTEGER PROCEDURE CLOSETANK;
    LIBRARY TANKLIB(ACTUALNAME = "CLOSETRBASE");

INTEGER PROCEDURE SEEKTRANSACTION(FILENUM, BLOCKNUM, OFFSET);
    INTEGER FILENUM, BLOCKNUM, OFFSET;
    LIBRARY TANKLIB;

INTEGER PROCEDURE READTRANSACTION (TRREC);
    TRANSACTION RECORD (TRB) TRREC;
    LIBRARY TANKLIB;



    TRANSACTION RECORD (TRB)
        TRIN,
        TROUT,
        RESTARTTR;
    INTEGER     IDNUM,          N,
                RESTARTNUM,     TANKNAMESIZE,
                FILENUM,        BLOCKNUM,
                OFFSET,         CT,
                TANKER,         UPDATER,
                MAXTANKER,      RSLT;
    ARRAY TRHISTORYUSERS[0:99];
    ARRAY SP[0:14];
    LABEL EXIT, LOOP, PRINTLAST;
    STRING ID, FNAME;
    BOOLEAN  ALLDONE;
    EBCDIC ARRAY SPO[0:79];

DEFINE ERR(L) =
       BEGIN
          REPLACE SPO BY "RSLT = ", RSLT FOR  * DIGITS,
                  "   ", LINENUMBER FOR 8 DIGITS, NULL;
          ACCEPT(SPO);
          GO L;
       END#,
       NULL         = 48"00"#,
       NORESTARTREC = 3#,
       REJECTED     = 2#,
       EOF          = 1#;


    MAXTANKER := 99;
    %
    %  SET THE LIBRARY PARAMETER "LIBPARAMETER" FOR THE TANK JOURNAL,
    %  AND THEN OPEN THE TRHISTORY JOURNAL FOR UPDATING AND THE
    %  TANK JOURNAL FOR READING.
    %
    SCAN TANKNAME[*] FOR N:99 UNTIL = 0;
    TANKNAMESIZE := 99-N;
    TANKLIBPARAM := STRING(POINTER(TANKNAME,8), TANKNAMESIZE);
    TANKLIB.LIBPARAMETER := TANKLIBPARAM;
```

```
IF RSLT := OPENTRHISTORY(1, 0) > 0 THEN ERR(EXIT);  % OPEN UPDATE
IF RSLT := OPENTANK(4, 0) > 0 THEN ERR(EXIT);  % OPEN FOR READING
ID := TANKLIBPARAM;
%
%  CREATE A USER OF THE HISTORY FILE AND THEN LOG HIM ON.
%
CREATETRHISTORYUSER(ID, RESTARTNUM);  % NO-OP IF NOT NECESSARY
IF RSLT := LOGONTRHISTORYUSER(ID, RESTARTNUM) > 0 THEN ERR(EXIT);
%
%  THE FOLLOWING CODE DETERMINES IF THE PROGRAM HAS BEEN RESTARTED
%  AFTER A HALT/LOAD. IF SO, IT DETERMINES THE LOCATION IN THE TANK
%  JOURNAL WHERE WE SHOULD BEGIN READING TRANSACTIONS. IT DOES THIS
%  BY EXTRACTING THE FILE, BLOCK AND OFFSET FROM THE ITEMS WITHIN
%  THE RESTART TRANSACTION RECORD: TANKFILENUM, TANKBLOCKNUM AND
%  TANKOFFSET. IF THE PROGRAM WAS NOT RESTARTED, THEN START READING
%  FROM THE BEGINNING OF THE TANK JOURNAL.
%

IF MYJOB.RESTARTED THEN
BEGIN
     REPLACE SPO BY "DETANKING PROCESS RESTARTING", NULL;
     DISPLAY(SPO);
     IF RSLT := RETURNRESTARTINFO(RESTARTNUM, RESTARTTR) =
                NORESTARTREC    THEN
     BEGIN
        FILENUM := 1;
        BLOCKNUM := OFFSET := 0;  % START AT FIRST RECORD OF FILE,
        IF RSLT := SEEKTRANSACTION(FILENUM,BLOCKNUM,OFFSET) > 0 THEN
           ERR(EXIT);
     END ELSE
        IF RSLT > 0 THEN
           ERR(EXIT)
        ELSE          % A RESTART RECORD EXISTS.
        BEGIN
           FILENUM := RESTARTTR.TANKFILENUM;
           BLOCKNUM := RESTARTTR.TANKBLOCKNUM;
           OFFSET := RESTARTTR.TANKOFFSET;
           REPLACE SPO BY "LAST GOOD TR FROM TANK AT (",
                FILENUM FOR  * DIGITS, ",",
                BLOCKNUM FOR  * DIGITS, ",",
                OFFSET FOR  * DIGITS, ")", NULL;
           DISPLAY(SPO);
           IF RSLT := SEEKTRANSACTION(FILENUM, BLOCKNUM, OFFSET) > 0
                                        THEN
              ERR(EXIT);
           %
           % NOW SKIP LAST GOOD TRANSACTION
           %
           IF RSLT := READTRANSACTION(TRIN) > 0 THEN ERR(EXIT);
        END;
END ELSE
BEGIN
     FILENUM := 1;
```

```
          BLOCKNUM := 0;
          OFFSET := 0;  % START AT FIRST REC OF FILE 1
          IF RSLT := SEEKTRANSACTION(FILENUM, BLOCKNUM, OFFSET) > 0 THEN
               ERR(EXIT);
     END;
     CREATE RESTARTTR.RESTARTDETANKER;

LOOP:
     IF ALLDONE THEN
     BEGIN
          CLOSETANK;
          CLOSETRHISTORY;
          GO PRINTLAST;
     END;

     %
     %  READ A TRANSACTION FROM THE TANK JOURNAL.
     %
     IF RSLT := READTRANSACTION(TRIN) > 0 THEN
     BEGIN
          IF RSLT = EOF THEN
          BEGIN
               ALLDONE := TRUE;
               GO LOOP;
          END  ELSE
               ERR(EXIT);
     END;
     %
     %  IF WE READ A SYSTEM TRANSACTION IGNORE IT AND CONTINUE WITH THE
     %  NEXT TRANSACTION IN SEQUENCE.
     %
     IF TRIN.TRFORMAT = TRFORMAT(SYSTEMTR) THEN GO LOOP;
     TANKER := TRIN.TRUSERNUM;
     IF TANKER > MAXTANKER THEN
     UPDATER := 0 ELSE
     UPDATER := TRHISTORYUSERS[TANKER];
     IF UPDATER = 0 THEN
     BEGIN
          IF RSLT := TANKUSERIDSTRING(ID, TANKER) > 0 THEN ERR(EXIT);
          CREATETRHISTORYUSER(ID, UPDATER);  % NO-OP IF UNNECESSARY
          IF TANKER > MAXTANKER THEN
          RESIZE(TRHISTORYUSERS[*], (MAXTANKER:=TANKER)+1, RETAIN);
          TRHISTORYUSERS[TANKER] := UPDATER;
     END;
     %
     %  SET UP THE RESTART RECORD VALUES TO BE THE ADDRESS OF THE INPUT
     %  TRANSACTION AND THEN PROCESS THE TRANSACTION.
     %
     RESTARTTR.TANKFILENUM := TRIN.TRFILENUM;
     RESTARTTR.TANKBLOCKNUM := TRIN.TRBLOCKNUM;
     RESTARTTR.TANKOFFSET := TRIN.TROFFSET;
     IF RSLT := PROCESSTRFROMTANK(UPDATER, TRIN, RESTARTNUM, RESTARTTR)
               > 0 THEN
```

```
          ERR(EXIT);
     GO LOOP;

PRINTLAST:
     FILENUM := RESTARTTR.TANKFILENUM;
     BLOCKNUM := RESTARTTR.TANKBLOCKNUM;
     OFFSET := RESTARTTR.TANKOFFSET;
          REPLACE SPO BY "LAST GOOD TR FROM TANK AT (",
               FILENUM FOR  * DIGITS, ",",
               BLOCKNUM FOR  * DIGITS, ",",
               OFFSET FOR  * DIGITS, ")", NULL;
          DISPLAY(SPO);

EXIT:
END OF THE DETANKER PROCEDURE;
```

D SYSTEMTR TRANSACTION FORMAT

One transaction format called "SYSTEMTR" is reserved for use by the system and is defined automatically by TFL in all transaction bases and subbases. Within the SYSTEMTR format, the following subformats are also defined:

REMOTEREQUEST

This subformat is used for communication between remote systems and the host system.

SYSTEMRESPONSE

This subformat is created by the Transaction Library whenever a response is required but not otherwise provided (for example, after a transaction processing error or during transaction tanking).

USERINFO

This subformat contains a brief summary of all user information contained in the control file. USERINFO is written at the beginning of each data file when a data file is created, and is written again when the last user calls the CLOSETRBASE Transaction Library entry point to close the transaction base. USERINFO is also used during recovery of the control file.

CHANGEUSER

This subformat is used to audit changes to the user information in the control file. CHANGEUSER is written in the data file when each change occurs (CREATETRUSER and PURGETRUSER), and is used during recovery of the control file.

A Series DMSII Transaction Processing System (TPS)

INDEX

Accessing Transaction Record Items, 88, 121
Addition, See Arithmetic Operators
ALGOL
   Detanking Procedure, 308
   Interface To TPS, 113
   Update Library, 303
   Use of TFL Items, 64
   User-Written Programs, 290
ALLINFO Option, 72
Alpha Item, 55
Application Languages
   and TFL Items, 64
AREAS, 46
AREASIZE, 46
Arithmetic Expression
   TFL, 23
Arithmetic Operators, 23

Begintransaction Statement, 180
BLOCKSIZE, 46
Boolean Expression
   TFL, 25
Boolean Item, 57

CARD File, 68
Case Rebuild, 232
Case Rollback, 232
Character Set
   TFL, 17
CHECKSUM, 47
CLOSEDATABASE Function, 171, 174
CLOSETRBASE Item, 159
COBOL
   Update Library, 299
   Use of TFL Items, 64
   User-Written Program, 279
COBOL/COBOL User Language Interface To TPS, 80A
COBOL74
   Specific Constructs For Parameter Passing, 105
   Use of TFL Items, 64
CODE File, 68
Comment
   TFL, 18
Compacting Data, 53
Compile-Time Functions, 100
Compiler Checks, 110, 136
Compiling The Remote Library, 194A
Compiling The Transaction Library, 139
Compiling Trutility, 234A
Computing Values For TRBITS, TRDIGITS, and TRBYTES, 102

Configuration
   of Remote Library Stack, 201
Control Items
   Transaction Record, 93
Controlinfo Statement, 250
Converting To Mixed Mode, 252
Converting To TPS, 251
CREATETRUSER Entry Point, 152
Creation Of Transaction Records, 86, 119

DASDL For Banking Example, 275
Data Base
   Invocation, 175
   Using A Single Library, 173
   Using Multiple Libraries, 175
Data Base Consistency And Update Library Programming, 184
Data Base Invocation
   Using A Single Library, 170
Data Compacting, 53
Data Item, 52
Data Item Qualification, 123
DEBUG Option, 72
DECKLIST Option, 72
Declaration Of Transaction Record Variables, 84, 117
Defaults, 33
DEPENDING ON, 53
Detanking Procedure
   ALGOL, 308
Detanking Procedure Example, 308
Digit
   TFL, 18
Division, See Arithmetic Operators
DMS Exception Handling, 218
DMSII User Language Syntax For Transaction Processing, 180
Dollar-Card Options
   and System/TFL, 69
DUPLICATED ON, 47

EBCDIC Characters
   and TFL, 17
Endtransaction Statement, 180
Entry Points, 150
Error Conditions, 199
ERROR LIMIT Option, 72
Example Of Detanking Procedure, 308
Examples Of Update Library, 299
Examples Of User-Written Programs, 279
Exception Handling
   DMS, 218
Exception Reporting, 203
Exceptions, 208

Expression
   Boolean, 25
   TFL, 22

FAMILY, 47
Family Name
   TFL, 19
Fatal Errors, 203
Field Item, 62
FORCEABORT Function, 171, 174
Format
   for Transaction Record, 37

Glossary, 258A
Greater than, See Boolean Expression
Group Item, 50

Halt/Load Recovery, 229
HANDLESTATISTICS Item, 158
HEADER Option, 73
HOST File, 68
Hostinterface
   and Remote Library, 194A
Hostinterface Program, 12

Identifier
   Simple, See Simple Identifier
INCLNEW Option, 73
INCLUDE Option, 73
Initialize Statement, 239
INITIALVALUE, 53
Input Files
   and System/TFL, 68
Integer
   TFL, 20
Interface To TPS
   ALGOL, 113
Invocation Of Transaction Base, 80B, 113
Invoking The Entire Database Using A Single Library, 170
Item
   Alpha, 55
   Boolean, 57
   Data, 52
   Field, 62
   Group, 50
   Number, 58
   Real, 60
Item Qualification, 123
Items, 49

Journal
   Reinitialization, 232
   Rollback, 231
   Statistics, 161
   Transaction, 44
Journals
   Transaction, 10

Language Syntax, 180
Languages
   and TFL Items, 64
LAYOUT Option, 74
Less than, See Boolean Expression
Letter
   TFL, 18
Library
   Compilation, 139
   Initiation, 142
   Remote, 12, 197
      Compilation, 194A
      Sample Stack Configuration, 201
   Transaction, 9, 139
   Update, 9
LINE File, 69
LIST Option, 75
LISTP Option, 75
LOGOFFTRUSER Item, 153
LOGONTRUSER Item, 153

MERGE Option, 75
Messages
   for Transaction Library, 160
Midtransaction Statement, 180
Mixed Mode Restrictions That Ensure Reproducibility, 252
Multiplicaton, See Arithmetic Operators

NEW Option, 75
NEWSOURCE File, 69
Not equal to, See Boolean Expression
Number
   TFL, 21
Number Item, 58

OCCURS, 53
OMIT Option, 76
Open Statement: TRupdate Option, 183
OPENDATABASE Function, 170, 173
OPENTRBASE Entry Point, 150

Operators
   Arithmetic, 23

PAGE Optioh, 76
Parameter Passing
   in COBOL74, 105
Parameter Passing To A COBOL Update Library, 168
Parameters, 30
   Transaction Record Variables As, 134
   Using Transaction Record Variables As, 104
PROCESSTRANSACTION Item, 154
PROCESSTRFROMTANK Item, 155
PROCESSTRNORESTART Item, 155
Program
   User-Written, 110
      Restart, 233
   User-Written Sample, 136
Programming The Update Library, 166A
PURGETRUSER Item, 152

Qualification
   of Data Items, 123

Railroad Diagrams, 254A
READTRANSACTION Item, 156
Real Item, 60
Record Variables
   Transaction
      Declaration of, 84
RecoverCF Statement, 247
Recovery, 225
   After A Halt/Load, 229
Reinitializing An Existing Journal, 232
Remark
   TFL, 21
Remopte Library
   Compilation, 194A
Remote Library, 12, 197
   Sample Stack Configuration, 201
Remote Library And Hostinterface, 194A
Reocrd Items
   Transaction
      Accessing, 88
Reproducibility, 191
Requirements For Data Item Qualification, 91, 123
Restart Of User-Written Programs, 233
Restrictions
   of Transaction Record Use, 109
   on Use of Transaction Records, 136
RETURNLASTADDRESS Item, 153

RETURNLASTRESPONSE Item, 154
RETURNRESTARTINFO Item, 154
Rollback of Journal, 231
Running Trutility, 237

Sample Remote Library Stack Configuration, 201
Sample User-Written Program, 110, 136
Search Statement, 240
Security, 223
SEEKTRANSACTION Item, 157
SEQ Option, 76, 77
SEQERR Option, 77
Simple Identifier
   TFL, 19
SINGLE Option, 78
SIZE of Alpha Information, 56
Stack Configuration
   of Remote Library, 201
Statistics
   Journal, 161
   Tank Journal, 161
   TPS, 160
STATISTICS Parameter, 31
String
   TFL, 22
Structuring The Update Library, 169
Subbase
   Transaction, 41
Subtraction, See Arithmetic Operators
SWITCHTRFILE Item, 158
SYMBOL/TFL, 14
System/TFL, 68
SYSTEM/TFL, 14
System/TFL Dollar-Card Options, 69
System/TFL Input Files, 68
SYSTEM/TRINTERFACE, 14
SystemTR Transaction Format, 314A

Tank Journal Statistics, 161
TANKTRANSACTION Item, 156
TANKTRNORESTART Item, 156
TAPE File, 68
TEXT Option, 78
TFL, 16A
   Basic Constructs, 17
   Boolean Expression, 25
   Character Set, 17
   Comment, 18
   Digit, 18
   Expression, 22, 23
   Family Name, 19

TFL (cont.)
   Integer, 20
   Number, 21
   Remark, 21
   Simple Identifier, 19
   String, 22
TFL Description Of Transaction Base, 28, 276
TFL Items in Application Languages, 64
TPS Recovery, 225
TPS Software Components, 14
TPS Statistics, 160
Transaction
   Definition, 7
   Limitations, 8
Transaction Base
   Invocation, 80B, 113
   TFL Description, 28
Transaction Base Example, 275
Transaction Bases
   Updating, 66
Transaction Compile-time Functions, 131
Transaction Compile-Time Functions, 100
Transaction Formatting Language (TFL), 16A
Transaction Journal, 44
Transaction Journals, 10
Transaction Library, 9, 139
Transaction Library Diagnostic Messages, 160
Transaction Record
   Control Items, 93
Transaction Record Control Items, 125
Transaction Record Format, 37
Transaction Record Items
   Accessing, 88, 121
Transaction Record Use Restrictions, 109, 136
Transaction Record Variable Assignment, 107, 134
Transaction Record Variables
   Declaration of, 84
   Declaration Of, 117
Transaction Record Variables As Parameters, 104, 134
Transaction Records
   Creation of, 86, 119
Transaction Recovery After A Halt/load, 229
Transaction Recovery After An Abort, 230
Transaction Recovery Design Features, 227
Transaction Subbase, 41
Transaction Use Procedures, 108
Transaction User
   Definition, 148
Transactions And Transaction Records, 7
TRBASE/HOSTINTERFACE, 14
TRBASE/HOSTLIB, 15
TRBASE/PROPERTIES, 15
TRBASE/REMOTELIB, 15

TRBASE/UTILITY, 15
TRBASEDTS Item, 127
TRBASEEDTS Option, 95
TRBITS
   Computing Values, 102
TRBLOCKNUM Item, 96, 127
TRBYTES
   Computing Values, 102
TRCOMPACTED Item, 97, 129
TRCONTROLSIZE Item, 95, 126
TRCOUNTER Item, 96, 127
TRDATASIZE Item, 96, 128
TRDIGITS
   Computing Values, 102
TRFILENUM Item, 96, 127
TRFORMAT Item, 96, 128
TRFORMATLEVEL Item, 95, 127
TRHISTORY Journal Statistics, 161
TRINPUTBLOCKNUMF Item, 98, 129
TRINPUTFILENUMF Item, 98, 129
TRINPUTOFFSETLOWR Item, 98, 129
TRINPUTOFFSETUPPR Item, 98, 129
TRLRBIT Item, 96, 128
TROFFSET Item, 96, 127
TRREJECT Item, 128
TRREJECTED Item, 97
TRREMOTE Item, 98, 130
TRRESTARTINFO Item, 129
TRRESTRARTINFO Item, 97
TRSIZE Option, 95
TRSTATE Item, 98, 130
TRSUBBASE Item, 96, 128
TRSUBFORMAT Item, 96, 128
TRTOTALSIZE Item, 95, 126
TRupdate Option
   for Open Statement, 183
TRUPDATEDTS Item, 127
TRUPDATEDTS Option, 95
TRUPDATELEVEL Item, 127
TRUPDATELEVEL Option, 95
TRUSERIDSTRING Item, 152
TRUSERNUM Item, 97, 128
Trutility, 234A
TRutility
   Statements, 238
TRUtility
   Compilation, 234A
   Running, 237

UPDATE Function, 170
Update Library, 9, 166A
   ALGOL, 303

Update Library (cont.)
   COBOL, 299
   Example, 177
   Examples, 299
   Programming, 166A
   Structuring, 169
Updating Transaction Bases, 66
User Language Interface To TPS, 80A
User-Written Program, 110
   ALGOL, 290
   COBOL, 279
   Restart, 233
USERCODE, 47

Variable Assignement, 134
Variables
   Transaction Record
      Used as Parameters, 104
Variables Assignement
   Transaction Record, 107
VOIDT Option, 78

WARNING Option, 79

ZIP Option, 79